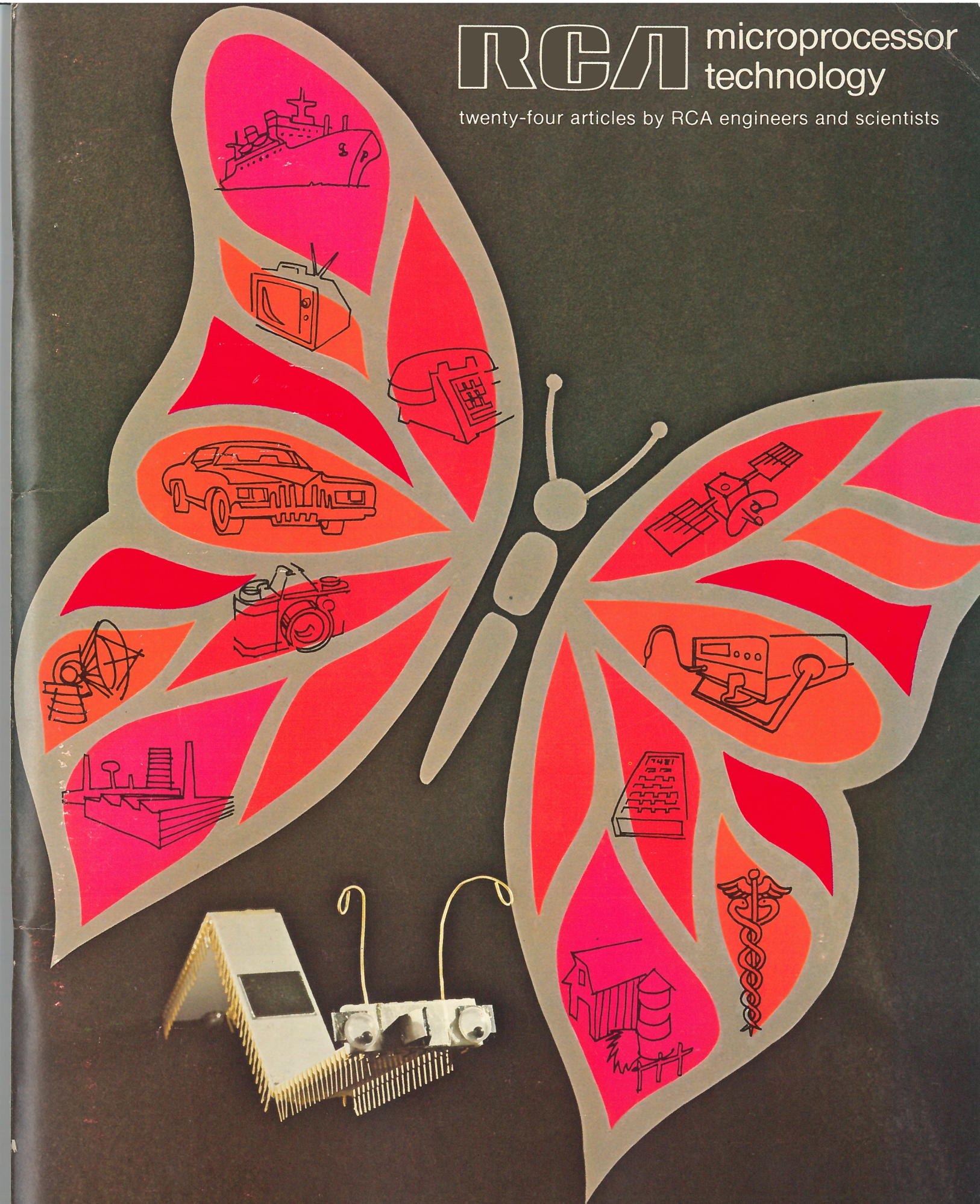


RCA microprocessor
technology

twenty-four articles by RCA engineers and scientists



Microprocessors—design in transformation

the microprocessor

technological change
economic opportunity

Today, the microprocessor is causing a major technological change in the electronics industry. We are now in the early phase of this change, which will eventually produce major market dislocations and significant leadership changes in various segments of the electronics market. The microprocessor's impact on the industry can be compared in magnitude to the conversion from tube circuits to solid state. At such a major point of inflection, opportunity is greatest for those who recognize the change and capitalize on it.

The RCA microprocessor program represents a success story; it was started in the RCA Laboratories in 1971, nurtured by the Solid State Technology Center through 1973 and 1974, and transferred to the Solid State Division in early 1975. It has now expanded to a full line of large-scale integrated circuits, complete with support hardware and software. The Solid State Division is solidly behind the microprocessor program, with strong continuing support from the Laboratories and the Solid State Technology Center.

The significance of this product to RCA, however, does not lie with the sales of the microprocessor circuits themselves. The real potential is in new electronic equipment products that creative RCA engineers can develop using this product line—new products fully exploiting the advantages of LSI economics coupled with computer architecture and software flexibility. This is the area where there is excitement and opportunity for both management and engineering within RCA to do something innovative in end products, to apply new design techniques, and to take a fresh unencumbered look at solutions to problems and needs.

In short, it is a time for RCA to use the cost-effectiveness and flexibility of microprocessors to seize the opportunity that is occurring in the industry today.



Philip R. Thomas
Philip R. Thomas
Division Vice President
Solid State MOS Integrated Circuits
Solid State Division
Somerville, N.J.

RCA microprocessor
technology

overview

R.O. Winder 2 The microprocessor business

introduction to microprocessors

J.C. Phillips 5 Getting involved with microprocessors
G.B. DiGirolamo 10 Learning to use microprocessors
W.D. Lauffer 13 Getting your hands on COSMAC hardware

COSMAC

A.W. Young 14 The CDP1802: a powerful 8-bit CMOS microprocessor
D. Block 20 A low-cost hardware prototyping system for microprocessor design
P.M. Russo 26 Architectural trade-offs in designing microcomputer systems
L.A. Solomon 35 COSMAC resident software development aids
C.A. Meyer 41 COSMAC support literature: keeping users informed
J.Weisbecker 45 Simplifying microcomputer architecture
J. Weisbecker 50 COSMAC VIP, the RCA fun machine
J. Weisbecker 55 A practical, low-cost home/school microcomputer system

advanced microprocessor technology

G.R. Briggs|S.J. Connor 63 20-MHz CMOS-on-sapphire COSMAC microprocessor
J.O. Sinniger|R.G. Stewart
W.A. Clapp|A. Feller 68 Technology impact on microprocessors
A.D. Feigenbaum|W.A. Helbig 72 The ATMAC microprocessor
S.E. Ozga

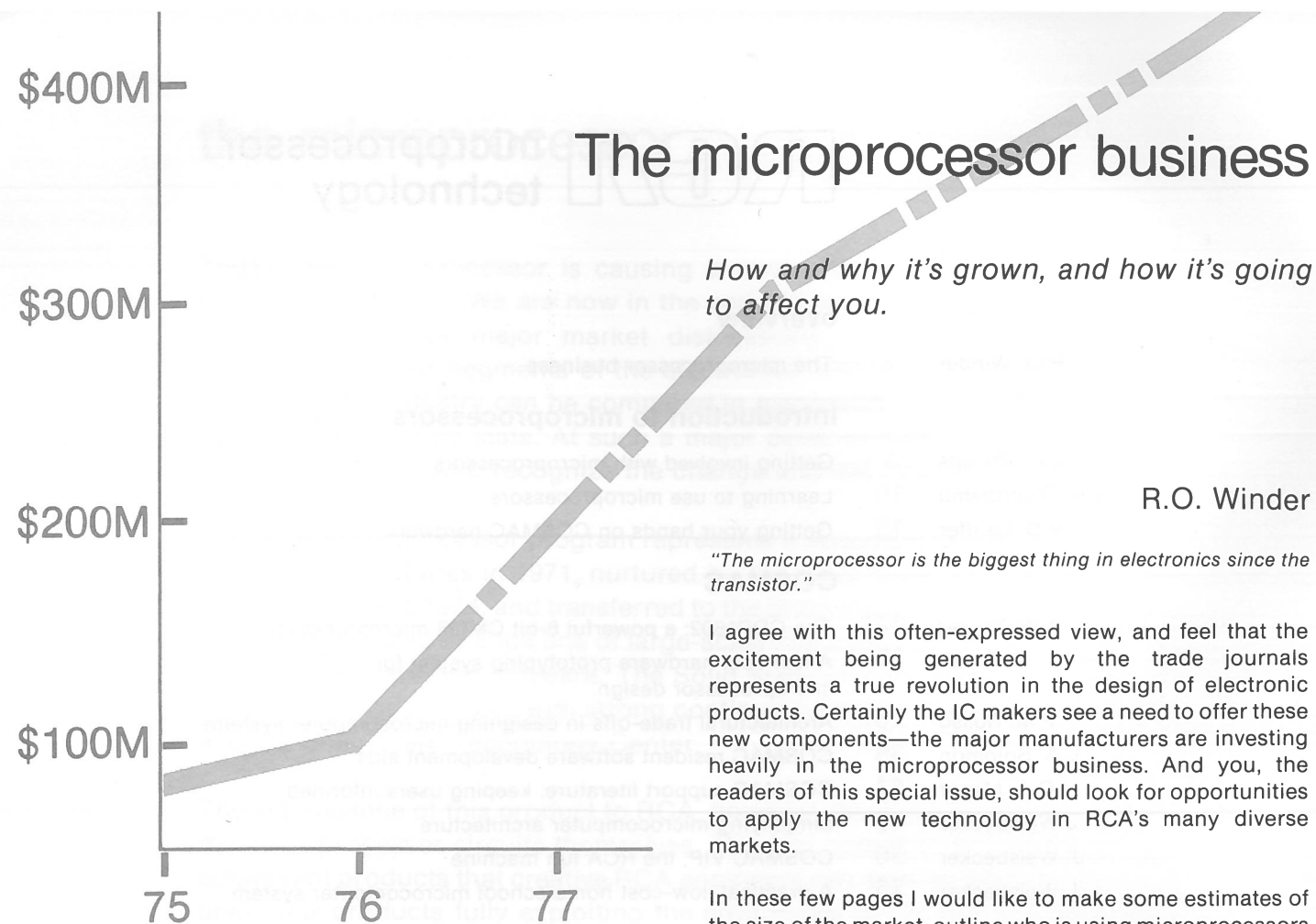
microprocessor applications

Engineering Research Notes 76 How RCA engineers are using microprocessors
M.D. Lippman 80 Microprocessors in telecommunications
C.C. Wang|C.T. Wu|P.M. Russo 84 Microprocessors in manufacturing and control
A. Abramovich|A.R. Marcantonio
P.M. Russo|A.R. Marcantonio 90 A microcomputer for test and control applications
C.T. Wu 94 CVM—a microprocessor-based intelligent instrument
J.D. Callaghan 98 Studio II—using COSMAC in a home entertainment product
W.M. Stobbe|W.M. Stonaker

automotive applications

A.D. Robbi|J.W. Tuska 101 A microcomputer-controlled spark advance system
W.R. Lile|J.W. Tuska 106 Electronic control of vehicle brakes
E.F. Belohoubek|J.M. Cusack 109 Microcomputer control for the car of the future
J.J. Risko|J. Rosen

115 COSMAC dictionary



Robert Winder is Director of Memory and Microprocessor Products in RCA's Solid State Division. His work with microprocessors began in 1971 at the RCA Laboratories, continued in the Solid State Technical Center in 1973, and moved to SSD in 1975. Dr. Winder received the David Sarnoff Award for Outstanding Technical Achievement in 1976 for his work with RCA's microprocessor products.

Contact him at:
Memory and Microprocessor Products
Solid State Division
Somerville, N.J.
Ext. 6005



What the products are

Semiconductor memory circuits were the first general-purpose large-scale integrated (LSI) circuits. They, together with the more specific calculator, watch, and tv-game circuits, plus numerous custom chips, take advantage of LSI technology to provide greater function, miniaturization, and reliability, at lower cost. Microprocessors, which contain the CPU (Central Processing Unit) of a computer on one (or a few) ICs, may be thought of as complements to memories, so when Intel introduced them, it meant that LSI could be used for the entire "main frame" of a computer—that part of a computer common to all applications. Even though customers had to provide SSI and MSI logic for interface to the outside world, the resulting systems had great advantages in flexibility, miniaturization, and cost—the revolution was underway.

Solid State Division's offering in standard memory parts includes CMOS, CMOS/SOS, and NMOS RAMs (random-

Final manuscript received October 18, 1976.

Table I
Rapid market expansion for the microprocessor families will be even more rapid in the next few years. (Source: *Electronics*, Jan 6, 1977.)

(millions of dollars)	1975	1976	1977	1980
Integrated circuits, total	1086.2	1512.7	1927.3	3106
Standard logic families, total	422.1	581.0	728.2	922
Microprocessor families, total	70.9	108.2	188.2	438
CPUs, total	24.9	38.5	63.9	166
MOS	23.1	35.6	58.9	150
4-bit	13.7	16.6	22.8	26
8-bit	5.8	13.0	25.0	70
16-bit	1.6	3.0	7.6	50
1-chip controllers	2.0	3.0	3.5	4
Bipolar, total	1.8	2.9	5.0	16
Bit slice	1.8	2.5	4.0	8
Full CPU	—	0.4	1.0	8
ROMs	14.5	16.7	32.0	80
RAMs	8.0	15.0	30.5	79
I/O interface chips	10.0	21.0	39.3	80
Peripheral chips	13.5	17.0	22.5	33

access memories) in 32x8, 256x4, 1024x1, 4096x1, and 1024x4 configurations, and CMOS ROMs (read-only memories) in 512x8 and (soon) 1024x8 configurations. Solid State Division does not presently offer a PROM (programmable read-only memory); their customers buy competitors' PROMs if their volume (under 50 or so) is too low for hard-wired ROMs.

Solid State Division's microprocessor business is built around the CDP1802 microprocessor, a CMOS implementation of Joe Weisbecker's COSMAC architecture, which is radically different from the Intel and Motorola architectures. COSMAC was specifically developed to minimize logic complexity, allow very compact programs, and interface efficiently with the outside world. This lower complexity permits us to manufacture the CDP1802 in CMOS at a cost comparable to NMOS and PMOS competition with its more complicated logic. And Solid State Division is able to compensate for its late start in this business by capitalizing on the well-known electrical benefits of CMOS technology—low and flexible power requirements, unexcelled noise immunity, and tolerance to wide temperature extremes.

The four-chip computer

As the market has grown, other kinds of product have become increasingly important—the circuits that interface the microprocessor to the outside world. These I/O (Input/Output) circuits consist of a mixture of custom and standard circuits. A typical custom I/O circuit would contain all the interfacing logic for the microprocessor, including A/D inputs, real-time clocks, and output-frequency generators. A customer would then have a 4-chip computer: 1-chip CPU, 1-chip RAM, 1-chip ROM, and 1-chip I/O circuit.

Such systems will sell for less than \$20 in 1978 in automotive volumes. However, not everyone buys in automotive volumes. Most customers will be using "I/O ports" to pass 8-bit data to and from the CPU, UARTs (universal asynchronous receiver-transmitters) to pass serial data, MDUs (multiply-divide units) to do fast arithmetic when required, and a collection of "connective tissue" parts to facilitate memory and I/O expansion. Solid State Division presently offers a simple byte I/O port, and the other parts listed are in various stages of development.

The one-chip computer?

Finally, I will point out that the virtues of LSI suggest pulling the various pieces of a microcomputer into common ICs—CPU and memory together, or ROM and RAM together, or even CPU, ROM, RAM, and I/O together. These products lose flexibility, but gain in cost advantages. As indicated in the market-summary table, they will gain in importance as LSI technology matures.

What they're used for

The market-forecast table lists the estimated size of the total digital IC market. In my opinion, the percentage growth of microprocessor products within this market represents the combination of two important trends: 1) microprocessors are displacing other digital markets, and 2) they are creating their own.

First, an increasingly higher number of digital systems that might have been implemented using SSI and MSI will instead be done in LSI—using microprocessor technology. In other words, what might have been an *ad hoc* collection of event-recognition circuits, counters, shift registers, miscellaneous imbedded RAMs, A/D converters, and messy

control logic will become organized into an efficient computer structure instead.

Why this trend? Because the microprocessor design process is better organized, more modular, more likely to implement the desired function, and generally is better understood. The various I/O interfaces to a computer are separate and work through a common bus structure and control scheme with the CPU. This makes deletions, modifications, and I/O additions easier. Control algorithms are implemented in subroutine-organized software—again, easier to modify. The payoff is in less engineering to reach a better product that is more easily maintained and enhanced.

The second trend swelling the market for microprocessors is the lower cost of LSI logic, which makes new products possible. For example, the automobile engine controllers that have been mechanical up to now—distributors, carburetors, etc.—will be electronic in the future. Computers will become commonplace in the home, beginning as tv-game add-ons. And LSI technology will accelerate the use of electronics in business and industry with new low-cost microcomputer-based products.

What will be left to SSI, MSI, and non-computer-like logic?

First, some I/O interfaces will never be offered as off-the-shelf products—they will have to be built up with the standard logic building blocks. Second, some systems are so simple that the “overhead” of a computer structure is too expensive. Third, there is a clear performance limit to what microcomputers can do. They are basically sequential devices. For a given stage of a technology, *ad hoc* logic—the “messy” printed circuit board full of standard logic circuits—can do many functions in parallel that will overtask a microprocessor trying to perform the same functions sequentially using that same technology. Although multiple microprocessors can be used, cost goes up. And fourth, for very-high-volume applications, certain functions will be done in custom LSI in an *ad hoc* manner (generally because they require high-speed parallel information processing). But the central core of digital systems will need a reasonable amount of “intelligence” and can be operated by one sequential processor, so that the microcomputer approach will be the best, whether it is done with custom microcomputers, with assemblies of standard microprocessor components, or with a single-chip microcomputer.

How they're supported

The micro-makers recognized early that their customers needed more help in using microprocessors than they needed with SSI, MSI, or memories. Few of the early producers had the benefits that RCA has: an in-house system capability, and even a few computer buffs left over from the good old days. Early support systems were primitive and were redesigned repeatedly. Applications-engineering help, always in short supply as the business grew, took some time to mature in quality. Early documentation was atrocious, and training seminars nonexistent.

RCA Solid State Division customer support is typical of that

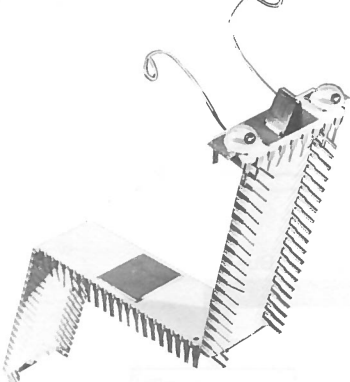
now offered by micro-makers. Support systems we sell include:

- the Microtutor, a minimum complete computer used for learning the basics of microprocessors;
- the Evaluation Kit, (see Block, p. 22) a PC board and package of parts sufficient to put together a small computer for electrical experimentation and prototyping (add only a power supply and terminal);
- the COSMAC Software Development Package, (see Solomon, p. 37) a FORTRAN program that provides an assembler, simulator, and interactive debugger (available as card deck or tape, or on any of several time-shared systems, including CMS in Cherry Hill);
- the COSMAC Development System, a complete, expandable rack-mountable microcomputer (just add terminal), with empty slots to support prototyping or small production volumes, software for assembly and editing, a floppy-disc option for fast and convenient software development, and a higher-level-language operating-system option for the most efficient design and debugging of microcomputer hardware and software;
- and (early in 1977) the RCA version of Intel's “in-circuit emulator,” a microcomputer that monitors and controls the activity of prototype or production microcomputer products—in short, the microprocessor engineer's oscilloscope.

Data sheets, manuals, application notes, seminars, and application engineering consultation—by phone and by direct visits—are an essential part of the support offered by all micro-makers (see Meyer, p. 43). Your access to application engineers will be a function of the supplier you work with and your importance to them. I'd like to say, on that subject, that you RCA customers, no matter how small the volume you need, are extremely important to us in Solid State Division. This is not only because of the obvious image impact, but because we believe we have the best products available, and that it is in RCA's best interest that you use them!

A final, especially interesting area of support the micro-makers offer is custom engineering for fee. This work ranges from implementing ROM patterns, through the design of custom I/O circuits, and into complete system design and software implementation. The Solid State Division, using RCA's unusual system-design capabilities, has done very well with this kind of service, getting prospective customers into production quickly with products likely to be successful.

The Advanced Technology Laboratory and the the Solid State Technology Center's Design Automation group provide an essential set of tools for these jobs—APAR (Automatic Placement and Routing) for fast turnaround custom I/O chips, and a ROM design system that generates IC masks and test programs automatically. I strongly recommend that RCA engineers working in microprocessors acquaint themselves with these capabilities—it costs suprisingly little to generate ROMs and custom ICs quickly. The SSTC will be happy to work with you in putting its arsenal of tools to work.



J.C. Phillips

If the lessons of history and present trends mean anything, the electronics world is going through a digital revolution—and the microprocessor is a primary catalyst. Digital technology—and specifically microprocessor technology—has grown so fast that many of you may still be groping for a place to start.

Start here. Those of you who have little or no knowledge of microprocessor technology should be able to pick up some of the buzzwords from this article—at least in sufficient depth to comprehend the message of this special issue, which focuses on the importance and the versatility of the microprocessor.

What is a microprocessor?

A microprocessor is the central processing unit of a microcomputer, and a microcomputer is simply a computer constructed from a handful of integrated circuits. Thus, any discussion of microprocessors and microcomputers can start with some general remarks about computers.

Getting involved with microprocessors

Start with this article if you think that every computer is bigger than a breadbox.

All computers perform five rather human-like functions: input, storage, control, processing, and output; Table I outlines these functions for humans and computers, including microcomputers. Although Table I is general, it places the control and processing functions in proper context with the input, storage, and output functions.

All microcomputers are based on these building blocks, with information exchanged among the blocks via a group of wires (one for each binary digit) called a *bus*; see Fig. 1.

How information is handled

The basic unit of information in any computer is the binary digit, or *bit*, capable of only two stable states, called “one” and “zero.” All computers (including microcomputers) operate on groups of bits called *words*. The word is the primary group of bits that the computer manipulates (reads in, stores, reads out, etc.) in a single step. *Word length* for any given computer is usually determined by the number of bits that can be stored in a specific memory location;

Table I
All computers perform these five functions. These are known as *Harvard-class* machines. If, in addition to these properties, the machine stores instructions in the same form as data in memory—each equally accessible to the calculating section of the computer—then instructions may be treated as data, and the machine can modify its instructions. Such a machine is called a *von Neumann* or *Princeton-class* computer. Microcomputers are available in Harvard and von Neumann classes.

Function	Human		Computer (microcomputer)	
	Device	Action	Device	Action
Input	Five senses (maybe six)	Receive sensory data and convert it to impulses for brain	Keyboards, magnetic tape, paper tape, punched cards, switch closures A/D converter	Receive data as binary “ones” and “zeros”
Storage	Brain	Store facts in brain for recall and reaction.	Tapes, discs, drums, cores, solid-state circuits	Place data (1s & 0s) in specific locations for later use.
Processing	Brain	Interpret data and instructions stored in memory.	Central Processor Unit (microprocesor)	Gather information from appropriate locations in memory (storage) and interpret results.
Control	Brain	Select appropriate action based on data and instructions.	Central Processor Unit (microprocessor)	Select an alternative course of action on the basis of computed results.
Output	Voice, actions, gestures, etc.	React	Printer, keyboard, switch closures, CRT, D/A converters, alphanumeric displays	Deliver results for user.

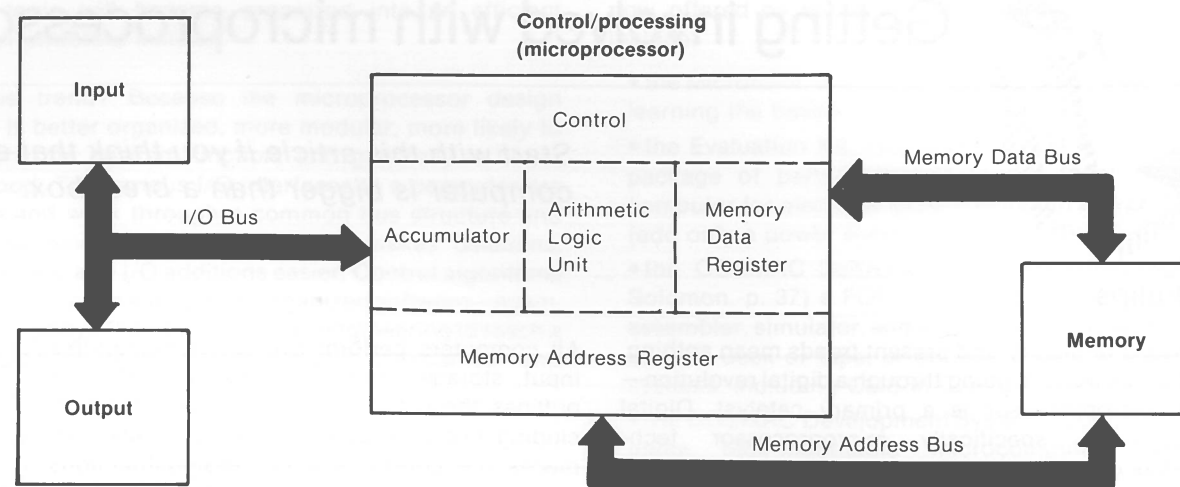


Fig. 1
Microcomputer architecture is based on four basic building blocks: input, output, memory, and microprocessor. The blocks are interconnected by buses—groups of lines [one for each binary digit (bit) to be transferred].

common word lengths are 4, 8, 12, or 16 bits. In general, the longer the word length, the greater the precision (number of significant digits). Also, the longer the word length, the richer the instruction set and the more varied the addressing modes.

An instruction set is the set of general-purpose instructions available with a given computer; generally, different computers have different instruction sets. Addressing modes are the ways that a given computer will specify the memory location containing the word to be operated upon. These modes are important factors in programming (and hence, in computer efficiency).

Two types of words are used in every computer: data words and instruction words. Data words contain the information that the computer processes; instruction words cause the computer to execute a particular operation. For example, an instruction may move data, do arithmetic and logic functions, control input or output (I/O) devices, or decide which instruction to execute next.

Byte is another commonly used term that is sometimes confused with word. A byte is any sequence (regardless of machine architecture) of n bits operated on as a unit. The most frequent size is 8-bits. A half-byte (4-bits) is called a nibble.

The basic task is to fetch and execute instructions

All of the operations of a computer are synchronized by a clock, and usually several clock pulses are required to accomplish the tasks specified by a single instruction. The execution of that single instruction is called an instruction cycle. For the RCA COSMAC microprocessor, a typical instruction cycle requires 16 or 24 clock pulses; a machine cycle requires eight clock pulses (see Fig. 2). Each instruc-

tion cycle typically consists of one or more machine cycles; the machine cycle is the basic microprocessor operation cycle. It usually consists of one of the following operations:

Fetch: The microprocessor uses an address (a number identifying a specific memory location) to read an instruction word from memory for use by the microprocessor; or

Execute: The instruction that has been fetched is decoded and the requested operation is performed.

There are other types of machine cycles (e.g., some types of interrupts), but most either fetch or execute.

Since a microprocessor must have sufficient hardware to fetch and execute instructions, it must contain:

- a program counter, which specifies the address of the next instruction to be fetched and executed. Normally, the counter is incremented automatically each time an instruction is fetched.
- an instruction register, which is a fast-access circuit used to store bits or words.
- an arithmetic logic unit (ALU), which executes such binary transformations as add, subtract, shift, AND, OR.
- a control section, which provides a way to address memory and to perform branch instructions (alter the sequence in which instructions are fetched based on the results of ALU operations).

Memory stores data and instructions

Instructions and data must be stored in memory so that they can be recalled at the appropriate time for the microcomputer to perform its function. Each instruction or data word is assigned a unique address that the microprocessor refers to when fetching or storing information. Memory is broadly divided into two classes: read-only memory (ROM); and read/write or random-access memory (RAM).

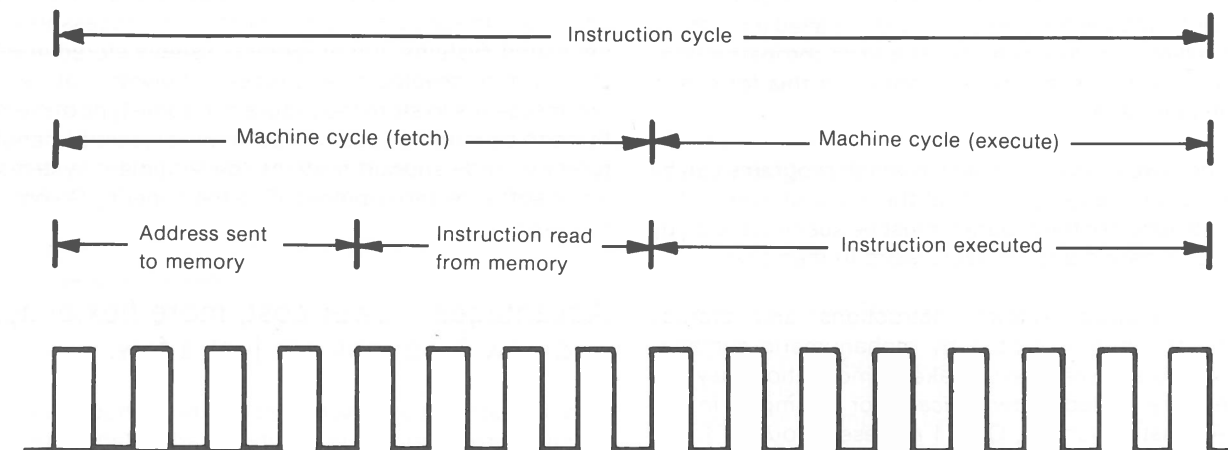


Fig. 2
Instruction cycle for COSMAC consists of one or more machine cycles; a machine cycle requires 8 clock pulses.

ROM (fixed memory) is used to store program steps and constant data values. It is difficult and time-consuming (and often impossible) to write into ROM; therefore, it is used in situations where the memory values do not change, e.g., in dedicated systems. Since this is a primary application of microcomputers, most of these machines use ROM for program storage. Some microcomputers use only ROM for storing program steps.

RAM (random-access memory) has both read and write capability. RAM is used to store data that changes during the operation of the system, e.g., results of calculations, or programs that are changed frequently.

Although most of the memory in a microcomputer is on separate chips, the microprocessor itself contains several fast-access storage units, called registers, that are used for temporary storage of data and instructions. The common registers are:

- Memory address register (MAR), which holds the address of the memory location being accessed.
- Memory data register (MDR), which accepts and temporarily stores the word coming from, or going to, memory (during read and write operation, respectively).
- Accumulator (AC), which holds the result of an arithmetical or logical operation to be used by the microprocessor. For example, the AC can provide an input to the arithmetic logic unit—ALU.
- Program counter (PC), which is automatically incremented to identify the next instruction to be fetched or executed.
- Stack pointer, which identifies (addresses) the last element in a stack. The stack is an array in RAM, separate from the microprocessor, which allows words or addresses to be accessed in a last-in, first-out fashion. A LIFO array is sometimes referred to as a pushdown array. Typically, a stack is arranged similar to a stack of papers, with each new input placed on top of the pile.

• Scratch-pad memory, which consists of general-purpose registers (RAM) that temporarily store data and addresses. The number and flexibility of such registers varies greatly among microprocessors.

• Instruction register (IR), which contains the instruction that has been fetched from memory and is being decoded and executed. This register usually has some external method of control (e.g., reset pushbutton or toggle switches) since the instruction register is typically used to direct the microcomputer to the location of the first program step.

• Status register (SR), which consists of one or more flip-flops (flags), provides information that is crucial to many arithmetic operations (for example, overflow from operations, zeros in the accumulator, sign of a number in the accumulator). This information is often used to decide what program step comes next.

Software —instructions that tell the microcomputer exactly what to do.

All operations that the microcomputer performs must be broken down into a series of individual tasks called instructions. The details and number of these instructions vary greatly depending on the microcomputer. Theoretically, any program may be written using the instruction set of any microcomputer. The length of the program and the time of the execution, however, may differ greatly with different instruction sets.

The instructions used may be divided into five functional types: transfer of data, control, subroutine linking, operation, and input/output. Such instructions may reference data from memory or from microprocessor registers, or may simply control the operation of the machine. Those that reference memory require an extra memory cycle to obtain the data and, therefore, may require more time to execute. In COSMAC, all instruction executes are identical.

A sequence of these detailed instructions needed for the computer to accomplish a specific task is called a *program*. Since the memory can store only 1s and 0s, the instructions must be encoded in binary. A program in this form is in *machine language*.

This is the lowest-level language in which programs can be written. Using it is tedious in that the value of every bit in every instruction of the program must be specified (e.g., by a string of binary digits for every word in memory).

Symbolic language allows instructions and storage locations to be represented by alphanumeric symbols. These symbols are chosen to make memorization easy and are therefore called *mnemonics*. (For example, in the COSMAC instruction set, OUT 1 represents *output 1*, IDL stands for *idle*, and SHLC means *shift left with carry*.) A program written in symbolic language must be translated to machine language before it can be stored and executed. The translation is performed by an *assembler* program. Most statements written in symbolic (or assembly) language translate to only one instruction in machine language.

To make programming easier, *higher-level languages* have been developed in which statements more clearly resemble English and mathematics, mostly for use on large-scale computers. Each is developed with a particular class of problem in mind. Some of the more popular languages are Fortran, ALGOL, APL, BASIC, COBOL, and PL/1. In these higher-level languages, one statement may correspond to many machine-language instructions. The conversion from a higher-level language to assembly code or to machine code is performed by a program called a *compiler*. (In a sense, a compiler is an assembler for higher-level languages.) Still another possibility is the popular BASIC, which is an *interpretive* language; i.e., the translation into machine code is accomplished at run time.

In preparing a program, it is frequently necessary to make numerous corrections and changes—for example, to correct a symbol or to insert one or more instructions into the program. This task is facilitated by the use of a special program called an *editor*. The editor allows the program to be stored into RAM and made available on an instruction-by-instruction basis for the purpose of making alterations easy.

Once written, a program is stored in one of the two types of memory. If the program is stored in ROM, it is always present in the machine and may be executed by branching to the address of the first instruction. This is the normal procedure in dedicated microcomputers. If the program is to be stored in RAM, which is volatile, then the program is usually loaded from keyboard, tape reader, or disc. To facilitate this loading, a special program called a *loader* is necessary and is often made resident in ROM. The loader is frequently combined with the capability to *read* or *write* directly into locations in memory so it can perform various housekeeping and debugging chores. In this case it is called a *monitor*.

Once a program is loaded, it is started by jumping to the first address in the program. Since most microprocessors are in dedicated systems, the program is usually stored in ROM. During the development process, however, it is very advantageous to store the program in some type of memory that can be modified easily. Most microprocessor manufacturers provide support systems (development systems) to aid in software development. (See the paper by Solomon in this issue.)

Advantages—lower cost, more flexibility, and new functions are just a few.

As demonstrated by several papers throughout this issue, the microprocessor offers significant advantages over both hardwired discrete logic and custom LSI (large-scale integrated) logic. The key advantage, of course, is that a given microprocessor chip can be used for a multitude of applications by simply changing the support hardware and the software. Thus, the cost of rather complex equipment is quite low because the microprocessor can be produced in large production volumes. In his excellent "Overview of microprocessor applications," in the June 1976 *Proceedings of the IEEE*, A.J. Nichols identified seven primary advantages of microprocessor-based designs, compared with discrete-logic designs.

- 1) The manufacturing costs of the product are lower. Typical microprocessor-based designs cost 20% to 60% of their TTL discrete-logic equivalents.
- 2) The time and cost for the original development is lower. An accomplished design team can cut the design time by about two thirds. As support tools improve, the design cycle will continue to decrease.
- 3) As a consequence, products can be brought to the market faster and in closer correlation with market needs. This can provide a significant edge in obtaining or increasing market share.
- 4) The microprocessor's inherent flexibility makes quick response to competitive pressures in the marketplace possible, with consequent increases in product lifetimes.
- 5) Greater functional capability can be provided at reasonable cost. This means better products for the same or lower prices.
- 6) The smaller number of components in a microprocessor system increases the reliability of the final product.
- 7) Should failure still occur, the microprocessor can, in some cases, perform self-diagnosis, providing substantial reductions in service charges.

In essence, the microprocessor offers the advantages of LSI without the cost penalty of designing the various mask and metalization patterns needed for custom LSI. Of course, high system complexity and significant production quantities will demand a custom design.

But there are some limitations

As stated previously, in doing its job of fetching and executing instructions by communicating with memory and with input/output devices, the microprocessor is a one-chip central processor unit for a microcomputer. However, placing all these central processor functions on a single integrated circuit imposes two very practical restraints:

- 1) The number of pins available for connection to the outside world—memory, I/O, power, and timing.
- 2) The number of functions that can be put on that chip.

The number of available pins limits the speed of information exchange.

Because the number of pins is limited, some type of serial (or multiplexed) data-transfer technique is used to carry information to and from the processor. In serial transfer, pins are traded for execution time; that is, entering 8 bits serially on a single pin takes 8 times as long as full parallel entry. Serial entry requires additional gating and storage circuits. Typically, memory interface pins are also used for I/O functions.

Functional limitations—designers are focusing on the rest of the system.

The number of functions that can be placed on a chip is limited primarily by the integrated-circuit technology used, and this limit is constantly changing (see the Clapp and Feller paper in this issue).

In general, several types of one-chip microprocessors have evolved to include the functions outlined above. Further work will probably be focused on improving the rest of the microcomputer system, with particular emphasis on input/output chips, and on multiprocessor systems (see the paper by Russo in this issue).

Implications for engineers—put your soldering iron on the shelf next to the tube tester and slide rule

In a sense, the microprocessor is bringing a breath of fresh air to the engineering lab. Probably the most tangible, albeit mundane, change brought about by the microprocessor is that the labs no longer reek of burning resin and carbon resistors. Breadboards are created in a new way—with manufacturer-supplied development systems (see the paper by Block, this issue). Design changes are implemented via software. Thus, learning about microprocessors also requires learning computer programming—the soldering gun and needlenose of the future!

RCA has made it easy for you to learn about microprocessors and computer programming (see the DiGirolamo paper in this issue). If you still don't think you should, reread Phil Thomas' inside cover message and glance at the market projections in Bob Winder's paper; also note the numerous applications cited in this issue.

Your efforts will be amply rewarded. A powerful, pocket-sized computer readily and economically available as a tool and as a design element carries significant implications for every engineer, regardless of discipline. There is a microprocessor in your future, and the future is now.

References

We asked several microprocessor experts to identify the microprocessor literature that may be useful and interesting—particularly to neophytes.

Mort Lewin of Rutgers University (consultant to the Solid State Division) called our attention to an exhaustive bibliography compiled by Ann R. Ward of Bell Laboratories. This bibliography appeared in two separate issues of the IEEE Computer Society journal *Computer* (part I in July 1974 and part II in January 1976).

Dr. Lewin then identified four introductory books with which he has had contact:

1. Hilburn, J.L. and Julick, P.M.; *Microcomputers/Microprocessors: Hardware, Software, and Applications* (Prentice-Hall; 1976).
2. Soucek, B.; *Microprocessors and Microcomputers* (Wiley; 1976).
3. Barna, A. and Porat, D.I.; *Introduction to Microcomputers and Microprocessors* (Wiley; 1976).
4. Adam Osborne and Associates, Inc.; *An Introduction to Microcomputers* (Berkeley, CA.; 1976).

Dr. Lewin particularly recommended the first and last, as did R. Smith from MSR, Moorestown, and several engineers from RCA Globcom. The Globcom engineers—John Frankle, Nick Giacketti, Bill Henn, and Tony Longo—also identified the following as useful sources:

5. Wester, J.G. and Simpson, W.D.; *Software Design for Microprocessors* (Texas Instruments, Inc.; 1976).
6. Motorola Semiconductor Products, Inc.; *Microprocessor Application Manual* (McGraw-Hill Book Co.; 1975).
7. *Proceedings of the IEEE*, Special issue on microprocessor technology (June 1976).
8. *EDN*, Special microcomputer reference issue (Nov. 20, 1976).

Specific microprocessor types are covered by the manufacturer.

Complete information on specific microprocessor types is usually available in the form of data sheets, instruction manuals, and application notes. For more information on COSMAC literature, refer to the paper by Meyer in this issue.

Applications areas papers are covered elsewhere in this issue.

Literature sources dealing with broad applications are listed with the papers in this issue that cover those application areas. For example,

Manufacturing and process control	Wang/Wu/Russo/Abramovich/Marcantonio
Test and control	Marcantonio/Russo
Communications	Lippman

Learning to use microprocessors

G.B. DiGirolamo

You can learn microprocessor technology and gain hands-on experience in microprocessor applications through a comprehensive course developed by Corporate Engineering Education exclusively for the RCA technical staff.

Engineers are generally aware of the enormous promise of the microprocessor, but many are unprepared technically and psychologically to use this tool effectively. In May 1975, Corporate Engineering Education introduced a course called "Microprocessors for Logic Design" that has helped more than seven hundred members of RCA's technical staff at seventeen locations understand microprocessor technology and develop "hands-on" skills in using microprocessors. "Hands-on" learning is built into the course through team projects using a microcomputer, the RCA COSMAC Microtutor. Several examples of in-course team projects recently completed by engineers at Automated Systems in Burlington, Mass. are shown below.

About the course

The objective is to teach microprocessor technology.

The course is intended to teach microprocessor technology in depth and to help develop skills in designing with, and for, microprocessor systems. Those who complete the course will be able

- To understand the internal structure of a microprocessor,
- To understand microprocessor interfaces with both random-access and read-only memories as well as input/output devices,

- To use microprocessors as a powerful tool in logic design.

Alumni of this course will also be able to do microprocessor system design—configuring a multiplicity of input and output devices and memories into a microcomputer. Finally, participants will learn the state-of-the-art in available hardware and software, be able to interpret manufacturers' specifications, and use effective criteria to evaluate competitive products.

As prerequisites, you should know something about logic design and computers.

To get the most out of this microprocessor course, participants should start with knowledge and experience in logic design equivalent to CEE course C30—Modern Logic Design I. Some exposure to programming and to computer concepts is helpful, but not essential.

The success of the microprocessor course depends heavily upon the effectiveness of an on-site instructor in dealing with the students' learning problems. Therefore, the course

also requires an Associate Instructor, who must have some prior microprocessor knowledge and understand computer fundamentals. Also he must know some assembly language, have written a program which has run successfully, and have logic design capability.

Optimum class size is sixteen participants.

Teams of four are formed in each class, and a Microtutor is assigned to each team.

Seventeen class sessions are suggested.

Included in the seventeen class sessions (2½ hours each) are fourteen video taped lectures, two sessions devoted to review and project demonstrations, and a session for the final examination. Homework is assigned between each session with "hands-on" team exercises between Sessions 5 and 9.

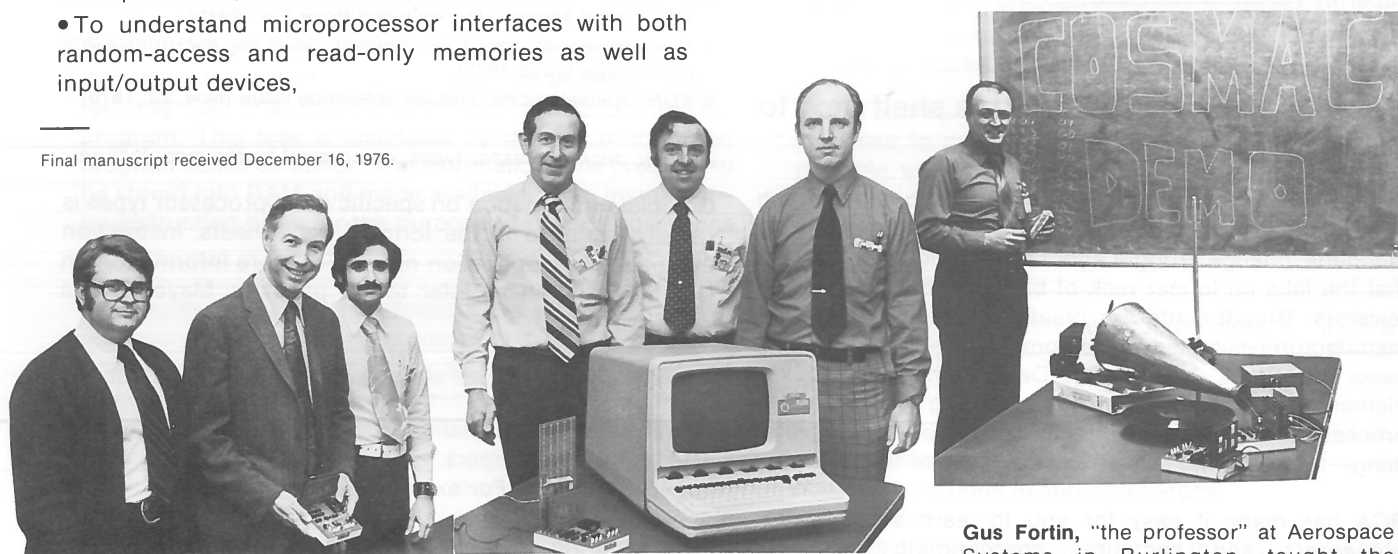
How to sign up for the course.

If you want to take this course, or any other CEE course for that matter, tell your supervisor or your local training manager. When a sufficient number of engineers (usually eight or more) express an interest, the training manager will arrange to get the course materials from Corporate Engineering Education.

Video-taped lectures, hands-on experience, and class participation are the important features.

The primary instruction medium is video-taped lectures. Each class session consists of the primary instruction supplemented by reviews of homework, class discussions, and class exercises, all led by the Associate Instructor.

A study guide, specially prepared for this course, contains printed forms of all the visuals and illustrations used on video tape. In addition, it contains several selected papers and reprints and represents a substantial presentation of



Final manuscript received December 16, 1976.

Jonathan Goode, Marc LeVarn, and Tony Macadino developed a "Random Number Game" in which a player attempts to correctly identify a random number generated by the microprocessor.

Ed Kramer, Roger Plaisted, and Ted Kupfrian decided to expand the limited input/output capabilities of the Microtutor by interfacing it to a video terminal as their project. They named their project "Microtutor 'UART'."

Gus Fortin, "the professor" at Aerospace Systems in Burlington, taught the microprocessor course and believes microprocessors are getting into all of our products, so he advises all designers to get aboard this new technology. As emphasized in the paper, the Associate Instructor is crucial to the success of this course.



Bill Shubert and Ron Tetrev devised a "Microtutor Editing Keyboard" using a hex keyboard and a two-character hex display to develop a unit to easily accept and modify machine-language programs.

Steven Hadden, Larry Hill, and Robin Hulls programmed music in a modern-day version of "His Master's Voice." Snoopy is listening to the "William Tell Overture" as programmed on the COSMAC Microtutor appropriately titled "Microtutor Music Master."

Where are all the others?

As mentioned in the article, about 700 RCA engineers have completed the CEE course, "Microprocessors for Logic Design," since its release in May 1975. This interest is indeed gratifying and actually not surprising in view of the excitement and importance of microprocessor technology. But 700 represents only a small fraction of RCA's engineers. How about you? Why haven't you signed up for the course?

Afraid it's too hot to handle?

Well it is tough. Requires a lot of homework. But think about the other tough things you've done. Why not accept it as a technical challenge that will be stimulating? Our very competent Associate Instructors will help you.

Figure you don't need it?

Come on now! How long can you really pass yourself off as a knowledgeable engineer without being conversant with one of the most significant technologies ever developed? How can you know you won't find an application for the technology when you really don't understand it? And isn't it just a matter of time? Why be last?

Maybe you'll learn it some other way?

Sure! It'll come as if by magic. You'll talk a little with this fellow and with that fellow, you'll read a few articles, and... Sound familiar? Don't count on it. This technology is more complex than that. Oh, you can learn some of the jargon, but let's see you write a program—even a simple one.

You're too busy?

Right! You've got all those commitments competing for your time. Don't we all! I bet every one of the 700 who've completed the course was just as busy. This is a matter of priorities. Think a little about the importance of microprocessors.

Don't have the prerequisites?

OK, that's legitimate. Why not get a group together and take our C30 course "Modern Logic Design." Then you'll have the prerequisite.

You'll wait for a better course?

There are all sorts of microprocessor courses available. Practically everyone in engineering education has one. But you'll have to go where it is and that won't be where you work. Also, it won't be tailored to RCA as this one is. And we doubt if you can find one as good as ours.

It's high time you think some about all this. Don't allow yourself to be obsolete in microprocessor technology. Now's the time to get in. It's going to become more difficult as time passes.

—W.J. Underwood

Bill Underwood is Director of Engineering Professional Programs and responsible for the Continuing Engineering Education function.

Contact him at: Corporate Engineering, RCA Bldg. 204-2, Cherry Hill, N.J., Ext. PY-4383

microprocessor technology. A syllabus of the course is given in Table I.

Extensive use is made of the RCA COSMAC Microtutor, which is a microcomputer designed by RCA Laboratories. As mentioned previously, this small device is used by class participants for "hands-on" experience. Participants write programs in assembly language and run them on the "Tutor." This activity has resulted in several interesting projects.

Future courses will deal with single-chip microprocessors and new applications.

Future course development, involving the COSMAC single-chip processor that replaces the two-chip version, is now being considered. As in the present course, the single-chip course would cover the hardware and software aspects of the microprocessor as well as the practical aspects. This includes the "hands-on" use of a Microtutor as an integral part of the course. More recent information on new applications of microprocessors and microcomputers will be included in the applied segment of the course.

The present course, C55, responded to a need and was successful because it was timely. This course brought a newly unfolding technology to the engineering work force. However, microprocessors differ from other technologies in that they put unprecedented power in the hands of the individual design engineer. The microprocessor course made this design tool understandable and useful.

Acknowledgment

Credit for the development and preparation of the microprocessor course is due Dr. Robert O. Winder, Solid State Division, Somerville; Dr. Anthony Robbi, Solid State Technology Center, Somerville; Dr. Paul Russo and Michael Lippman, RCA Laboratories, Princeton; Richard Smith, Missile and Surface Radar, Moorestown; and Dr. Morton Lewin, formerly with RCA Laboratories and now with Rutgers University. The author acknowledges with gratitude their contributions in making the microprocessor course possible.



Gerry DiGirolamo is the course development administrator for the microprocessor course described in this paper. He has been working in Corporate Engineering Education since 1965. In the photo, he is directing a video-taping session. Contact him at: Continuing Engineering Education RCA Bldg. 204-2 Cherry Hill, NJ Ext. PY-5141

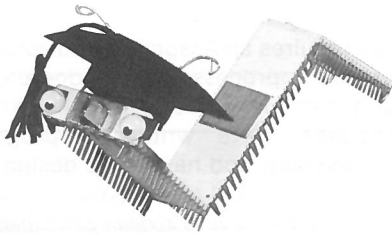


Table I
Microprocessors for logic design, course outline. Sessions 16 and 17 are for review and final examination, respectively.

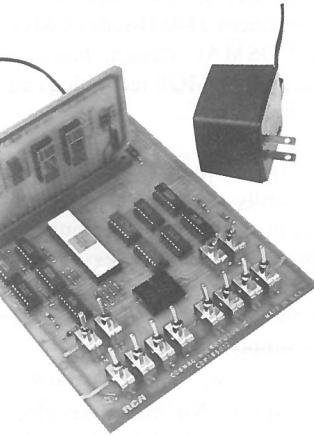
Session	Description
1 — overview	What a microcomputer is, and what a microprocessor is; why and when to use a microprocessor.
2 — basic processor operations	Binary numbers. Hexadecimal notation and the concept of a "byte." Random-access memory and its addressing. Read-write vs. read-only memories. Instruction format of COSMAC.
3 — COSMAC architecture I	COSMAC instruction actions, including the concept of the conditional branch. The input/output instruction.
4 — COSMAC architecture II	COSMAC binary arithmetic instructions. COSMAC arithmetic-logic-unit.
5 — architecture and assembly language	COSMAC architecture review. COSMAC instruction set. Assembler directives.
6 — stacks and subroutines	Subroutines. Nesting. Stacks. Interrupt service routine.
7 — software techniques and system development	Assembly-level programming. Trade-offs between assembly level and higher level languages. Time-sharing software development systems. Stand-alone microprocessor development systems.
8 — input/output structures	One-level I/O structure. Two-level I/O concept. Microprocessor-based system design. Operation of the DMA channel. External flags (EF1-EF4) to identify and decode interrupts. Interrupt priority.
9 — single-chip COSMAC microprocessor	Differences between the single-chip COSMAC and its two-chip predecessor. The 1800 family of support chips.
10 — midterm review and test	Review of sessions 1 through 9. Demonstration of team projects.
11 — comparing microprocessors	LSI-oriented technologies. Intel 8080 and the Motorola 6800. Comparisons with RCA COSMAC system.
12 — bit-slice processor	Bit-slice microprocessor. Monolithic Memories (MMI) 5701/6701. Application to coordinate conversion.
13 — RCA FRED system	FRED, an experimental model of a minimum-cost home/school computer. Use of an unmodified tv set as a display. The power and flexibility of the dot-matrix display philosophy.
14 — microprocessor-based data communications system	Narrowband dedicated leased-channel store-and-forward data communications system. The TTY and RS-232 communications interface. Floppy discs and floppy disc drives.
15 — microprocessor-based data communications systems and multiprocessor structures	Difference between bits/second and baud rate. Floppy-disc seek, and latency, access, and data transfer times. Worst-case and average disc thruput rates. Disc thruput rates, related to data communication system thruputs. The general network and master-slave organization. Two-microprocessor implementation of leased channel system.

Getting your hands on COSMAC hardware

Eventually, you're going to have to stop reading about microprocessors and start using them. Here are the COSMAC options available for when that time comes.

W.D. Lauffer

If you have no previous computer knowledge, Microtutor II is the place for you to start.



An improvement on the popular Microtutor I, it includes everything you need to write and run machine-language programs. You can learn to program games, educational exercises, and controllers quickly—the *Microtutor Instruction Manual's* lively style is an excellent example of making technical education painless.

Specifically, Microtutor II includes the CDP1802 microprocessor, 8-bit toggle-switch inputs, a crystal clock, a two-digit hex display, power supply, and a 256-byte RAM—all contained on two printed-circuit cards. There are also sockets for memory expansion and external options. For example, built-in operating system memory addressing is possible with only one more 256-byte RAM card.

Or, start from scratch with the COSMAC "ELF."

The true experimenter will want to start with the individual chips and components and build up a microcomputer system. If this is the route you'd like to take, consider the COSMAC "ELF." Joe Weisbecker has written a series of articles on its construction, programming, and expansion in *Popular Electronics*. The issues are August '76, September '76, and March '77; a future article will cover video interfacing.

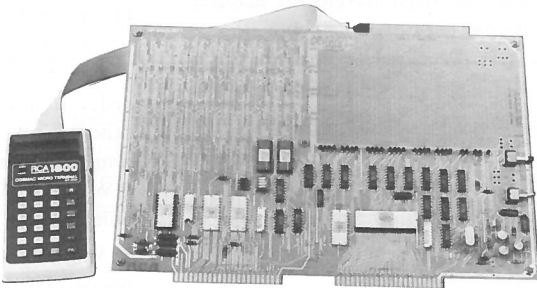
COSMAC VIP is designed specifically as a home computer.

Sometime this year, VIP will appear on the market. A low-cost home computer aimed at video graphics and games, it uses a video monitor, audio cassette recorder, and a 16-position key pad as I/O devices—keeping total system cost down. As designer Joe Weisbecker says, "It's not fair to ask the limited-budget, unsophisticated beginner to buy a system for several hundred dollars but then require an

additional \$1000 for a terminal and enough RAM to make it useful." These I/O devices and a hex interpretive language make is possible for users to write and load their own games and programs simply and inexpensively.

Compare COSMAC with the competition.

The COSMAC Evaluation Kit was designed to let you see just how RCA's 1800-series components work. The kit includes a printed-circuit board, a manual, and all the parts you'll need to evaluate, and design various systems around, the CDP1802 CPU, the CDP1832 ROM, the CDP1824 RAM, and the CDP1852 byte I/O port.



By adding a teletypewriter, hand-held Microterminal, or other terminal, you can use the built-in ROM utility routines to write programs and exercise the system. The board has room for memory and I/O expansion. For a detailed description, see Dennis Block's article on the Evaluation Kit in this issue.

Do system design with the COSMAC Development System.

When you get down to making prototypes, you'll need to breadboard and debug hardware/software configurations on the way toward the final design. This is where the COSMAC Development System (CDS) fits. Its 19-inch rack-mountable chassis provides a power supply, front-panel controls, printed-circuit backplane, a set of small printed-circuit boards, and 22 empty card slots. The basic set of cards can support up to 65,536 bytes of memory.

The CDS can be used with a number of different terminals ranging from paper tape to floppy disc; speed will depend on the investment you decide to make.

CDS software includes the COSMAC resident assembler and resident editor. Larry Solomon's paper in this issue should tell you how to go about using them while developing your COSMAC system.

The CDP1802: a powerful 8-bit CMOS microprocessor

A.W. Young

Learn the advantages of COSMAC and its supporting family of chips; then find out what's involved in putting a system together.

The RCA CDP1802 microprocessor is a powerful and efficient single-chip 8-bit CPU that forms the heart of a general-purpose microcomputer. It is easy to understand, easy to use, and low in cost. COSMAC architecture is based on the efficient use of short-word-length instructions within an interface restricted to 40 pins. Instead of using a minicomputer instruction set forced into 8-bit bytes with multiple-byte instructions, the COSMAC architecture has been developed around single-byte instructions.

Final manuscript received November 29, 1976.

Alex Young, as Manager of Memory/Microprocessor Design Engineering, is responsible for the design of the RCA1800 microprocessor family and its support systems. Previously, he had been Manager of COS/MOS Engineering and had designed over 20 CMOS circuits as a member of the Custom Circuit Design Group.

Contact him at:
**Memory/Microprocessor
Design Engineering
Solid State Division
Somerville, N.J.
Ext. 6468**

The applications addressed by the CDP1802 include low-end minicomputer replacement, general-purpose controllers, emulation of hardwired logic, and totally new areas made possible by the advent of low-cost, stored-program, general-purpose data processing.

The CDP1802 is fabricated with a self-aligned silicon-gate CMOS technology, giving these results:

- 1) *Full temperature range* (-55°C to $+125^{\circ}\text{C}$) for ceramic-packaged devices.
- 2) *Low power dissipation*—generally an order of magnitude or more less than other 8-bit microprocessors at maximum processor speed. Specifically, 50 mW at 6.4 MHz and 10 V; less for lower voltages and slower clock rates.
- 3) *Single noncritical wide power-supply range*. The CDP1802 will operate from 3 to 12 V over its full temperature range. Since the low-current single-voltage supply does not require sophisticated regulation, power-supply design is straightforward and simple.

The CDP1802 has captured these CMOS technology features to the extent that it "looks like" another CD4000-series CMOS device. The COSMAC design, however, provides additional CMOS technology advantages, including:

- 1) *High noise immunity*. Inputs have been specifically designed to switch at 50% of supply voltage. To account for normal processing tolerances and the effects of temperature and voltage range, the design point translates to a guarantee of 30% of supply voltage.
- 2) *Static operation*. The CDP1802 is completely static. No minimum clock frequency is required and no complex multiphase clock generators are necessary. An on-chip oscillator amplifier is provided and requires a single feedback resistor and crystal connection to operate the clock. Alternately, an external single-phase RC oscillator could be used as the system clock.
- 3) *High speed*. The CDP1802 will operate at 10 volts with instruction times

Table I
Features of the CDP1802 microprocessor.

Static silicon-gate CMOS technology—CD4000-series compatible	Simple control of reset, start, pause, and load modes
Instruction times of 2.5 or 3.75 μs at 10 V	8-bit parallel organization with bidirectional data bus
Compatible with CDP1801 software	Works with any combination of the standard RAM and ROM
Full temperature range (-55°C to $+125^{\circ}\text{C}$)	Memory addressing of 65,536 bytes
30% noise immunity	91 instructions
Single supply voltage	On-chip DMA
Wide operating voltage range: 3-12 V	Program interrupt
No minimum clock frequency	Programmed I/O
Low power—50 mW at maximum 10-V instruction time	Four input flags
TTL-compatible	Programmed output port (1-bit)
Single-phase clock	16 $\times$ 16 register matrix for use as multiple program counters, data pointers, or data registers
Optional on-chip crystal-controlled oscillator	

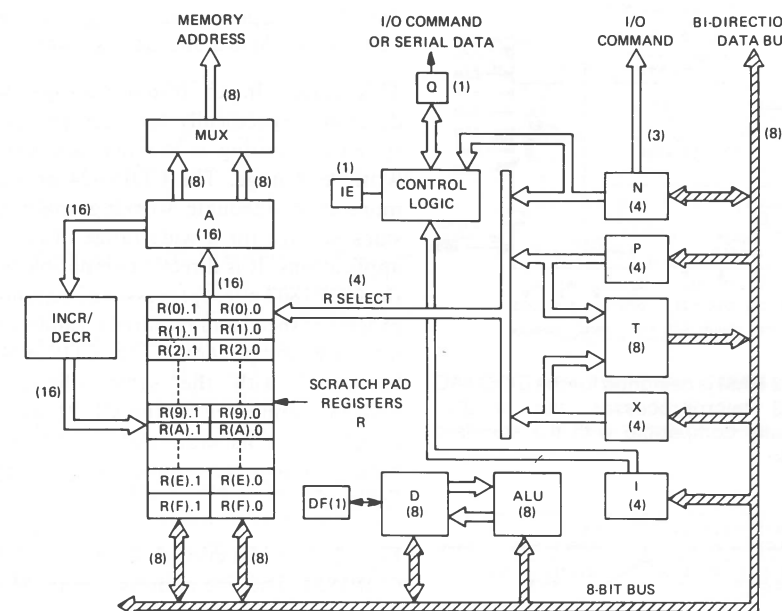


Fig. 1
COSMAC architecture is based on sixteen on-chip 16-bit scratchpad registers. Three 4-bit pointers (N, P, and X) can designate any of the 16 general-purpose registers as a program counter, memory address register, data source, or data destination.

of 2.5 or 3.75 μs . All but long-branch and long-skip instructions execute in 2.5 μs .

High output drive, low input current, low quiescent power, and extra static protection are also provided by design. The combination of COSMAC architecture and CMOS technology make the CDP1802 a high-performance cost-effective device for a broad range of applications. Specific features of the CDP1802 are summarized in Table I.

COSMAC architecture

The COSMAC architecture has been carefully developed for efficient operation in an 8-bit environment. Fig. 1 shows the internal processor architecture.

The basic architecture provides 16 on-chip 16-bit scratchpad registers.

These registers (R) can be used to point to data in memory, point at programs, or store data (two bytes per register). An additional 8-bit data register (D) is used to buffer data transfers between an R register and the data bus and also functions as a conventional accumulator.

Any of the 16 general-purpose registers can be designated as a program counter, a memory address register, a data source, or a data destination by means of one of three 4-bit pointers.

The register labeled P holds the 4-bit program-counter pointer. One instruction loads P with any 4-bit number, so the program counter can be changed by simply changing the number in P.

A second 4-bit register, N, stores a variable pointer carried along with the instruction. For example, a load instruction designates one of the 16 registers as an address register. The bits in the instruction doing the pointing are stored in N to designate the appropriate register.

A third 4-bit register, X, stores a pointer used to designate an address register during input/output and some ALU instruction. Like P, it can also be loaded with any number by a single instruction.

Using four-bit pointers to indirectly specify a 16-bit address is a key feature of the COSMAC architecture.

This feature results in compact code because operand addresses can be efficiently carried by a short 8-bit instruction. By providing a generous quantity (16) of general-purpose registers, the programmer can set up many pointers to data tables and utility routines and so access them quickly and efficiently.

The notation R(P), R(N) or R(X) refers to the general-purpose register designated by

the 4-bit code in P, N, or X, respectively. M(R(P)), M(R(N)), or M(R(X)) refers to the memory byte pointed to by R(P), R(N), or R(X), respectively. Using hex notation, R(3) designates the general-purpose register corresponding to binary code 0011. R(3).0 refers to the least-significant byte of R(3), and R(3).1 refers to the most-significant byte of R(3).

In addition to the general-purpose register array, data register, and register pointers, the COSMAC architecture provides a conventional ALU that performs arithmetic and logic operations between operands stored in D and M(R(X)) or M(R(P)), with the result stored in D. An overflow bit, DF, can be used for conditional branching.

Instruction cycles are subdivided into fetch and execute cycles, also referred to as machine cycles. When instructions are fetched from M(R(P)) during the instruction-fetch cycle, the four most-significant bits are placed in register I and designate an instruction op-code. The four least-significant bits are placed in register N and specify R(N) as a data destination (as in D $\rightarrow$ R(N).0—put low R(N)) or as a memory address register (as in load D with M(R(N))). The four-bit N field is also used to extend the op-code, as in ALU and branch instructions.

The COSMAC I/O interface has been given particular prominence in the architecture design.

This interface can be summarized as follows:

- 1) *Four input flags* ($\overline{\text{EF1}}$ – $\overline{\text{EF4}}$), which can be tested by conditional branch instructions.
- 2) *A serial output* (Q), which can be set and reset under program control and tested by conditional branch instructions.
- 3) *Programmed I/O data transfer*, which uses the N bits as a device-select code and transfers data between the selected device and memory.
- 4) *A maskable interrupt mechanism*, which can be activated by the $\overline{\text{INTERRUPT}}$ input. When this interrupt occurs, the old values of P and X are automatically saved in temporary register T and new values are jammed into P and X, effecting a 1-cycle switch to a new program counter for interrupt servicing.
- 5) *A DMA channel*, which can be activated by $\overline{\text{DMA IN}}$ or $\overline{\text{DMA OUT}}$

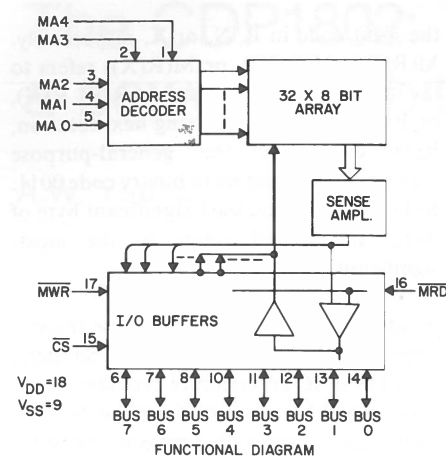


Fig. 2
Static 32-byte CMOS RAM, the CDP1824, was developed specifically for microprocessor systems requiring a minimal amount of writable storage.

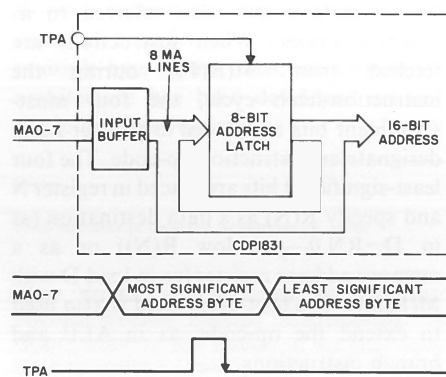


Fig. 4
Mask-programmable COS/MOS ROM, the CDP1831, uses address-multiplexing method shown here.

and uses R(0) as the DMA pointer. Each DMA request causes one machine cycle to be "stolen," appropriate memory address and control signals to be generated, and the pointer incremented.

6) **Timing synchronizing signals** (TPA and TPB), which assist in data transfers and general system timing.

The CPU performs instructions in 2.5 or 3.75 microseconds.

General operation of the CPU is controlled by the inputs WAIT and CLEAR, giving four possible operating modes:

LOAD—Holds the CPU in the idle state and allows an I/O device to load memory using R(0) as the memory address register.

RESET—"Resets" the CPU by forcing X, P, Q, and R(0) to 0 and enables interrupts. After RESET, the CPU starts execution of instructions from memory

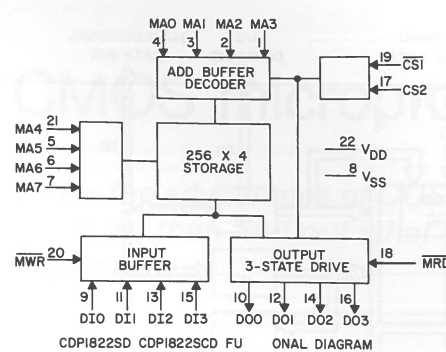


Fig. 3
CDP1822 RAM is designed for the COSMAC CDP1802 microprocessor, but is also functionally compatible with the standard 2101 type.

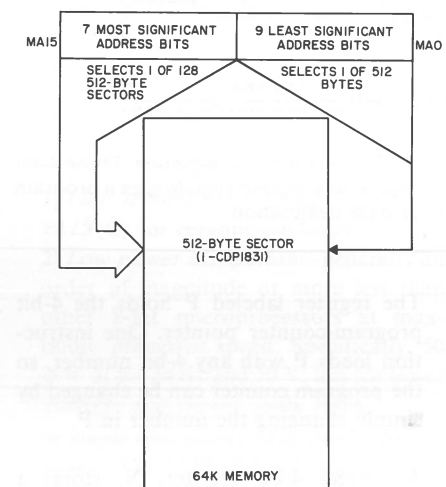


Fig. 5
Internal allocation of the 16 address lines in the CDP1831 ROM.

location 0000 with R(0) as the program counter.

PAUSE—Stops the internal CPU timing generator on the first negative high-to-low transition of the clock. The on-chip oscillator, if used, continues to run but is ignored.

RUN—Normal operation where the CPU fetches and executes instructions.

While in the RUN mode, the CPU can be in one of four states: instruction fetch; instruction execute; DMA; and interrupt.

All instructions require a fetch and one or two execute cycles. Since each cycle consists of 8 clock periods, instructions require 16 to 24 clock periods, which translates to 2.5 or 3.75 μ s at 6.4 MHz.

Support circuits

A full family of 1800-series support circuits, including memory and I/O, complement the CDP1802 microprocessor.

The RAM support circuits begin with the CDP1824 32-byte static CMOS RAM.

This device, in an 18-pin package, was developed specifically for microprocessor systems requiring a minimal amount of writable storage. The CDP1824 provides more than adequate working space and stack storage for a wide range of control applications. It is directly compatible with the CDP1802 microprocessor at maximum processor speed and requires no additional interface components. The CDP1824 is fabricated with the same silicon-gate CMOS technology as the CDP1802 and therefore has its same wide-temperature-range, low-power, and single-supply features.

Fig. 2 is a functional diagram of the CDP1824. The five address inputs, MA0-MA4, are buffered and decoded to uniquely select 1 of 32 bytes. The eight input-output data lines interface directly with the microprocessor data bus. Operation is determined by the state of the MWR and MRD inputs. For MRD = 0, the output drivers are enabled and the data stored in the addressed byte will be put on the data bus. For MWR = 0, the contents of the data bus will be stored at the addressed location. A chip-select input CS is provided to enable the memory.

The CDP1822 is a 256 x 4 static RAM.

It is functionally and pin-compatible with the 2101 industry type and is also pin- and performance-compatible with CDP1802. Fig. 3 is a functional diagram of the CDP1822. The eight address inputs, MA0-MA7, are buffered and decoded to uniquely select 1 of 256 four-bit words. The four data outputs are driven from the three-state drivers that are enabled by MRD when the device is selected. Normally, when used with the CDP1802, the four data-input and four data-output terminals are connected together and to the CDP1802 bidirectional data bus.

The CDP1822 requires no clock or precharge signals for proper operation and interfaces directly to the CDP1802 without additional components.

The CDP1831 is a static 4096-bit mask-programmable COS/MOS read-only memory organized as 512 8-bit words.

It interfaces directly with the CDP1802 without additional components. The CDP1831 responds to a 16-bit address multiplexed on 8 address lines (MA0-MA7), 8 bits at a time. Fig. 4 shows how the

address lines are multiplexed and latched to form the 16-bit address. The most significant address byte is latched internally by either a positive or negative user-defined level of TPA. For compatibility with the CDP1802, this level is negative, as indicated in Fig. 4.

Fig. 5 shows the internal allocation of the 16 address lines. The seven most significant address lines select one of 128 512-byte sectors of the 64-kilobyte microprocessor memory space. The sector address is equivalent to a ROM-select and is user-programmable for a specific CDP1831.

The seven-bit ROM-select code is "anded" with the three chip selects (CS1, CS2, and MRD) to enable a given ROM. This signal is provided on an output pin (CEO) of the CDP1831 to indicate when the ROM is enabled.

The nine least-significant address lines are decoded to select one of 512 bytes in the ROM (sector). In summary, the CDP1831 responds to a 16-bit address; this address specifies one of 128 memory sectors, which is fixed for a given ROM, and one of 512 contiguous bytes within the specified ROM.

There are three important features of the CDP1831 that result from this addressing structure. First, it is directly compatible with the CDP1802 multiplexed address bus. Second, it is possible to make up a ROM system of from 512 to 64k bytes that is directly compatible with the CDP1802 and requires no address decoding. Third, when the ROM is selected, the CEO output goes high and can be used directly to disable a RAM system.

Reading data out of the CDP1831 follows a standard procedure. The device is selected by clocking the proper sector-address into the address latch. The tri-state output buffers are enabled by the proper combination of the three chip-selects, CS1, CS2, and MRD. The chip-select enable polarities are user mask-programmable. Valid data will appear at the output within one access time (t_{AA}) from the last address change. Fig. 6 is a functional diagram of the CDP1831.

The CDP1832, like the CDP1831, is a static 4096-bit mask-programmable COS/MOS read-only memory organized as 512 8-bit words.

It is conventionally organized, requiring 9 address lines and a chip-select (CS) to read

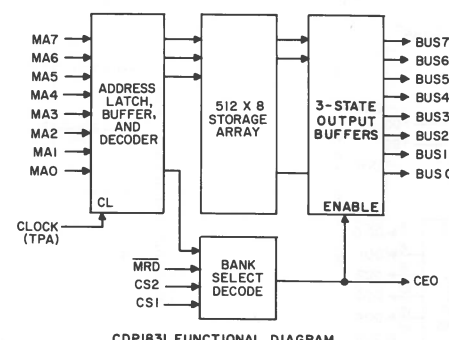


Fig. 6
Reading data out of the CDP1831 follows standard procedure to produce valid data within one access time from the last address change.

one of 512 eight-bit words into a bidirectional data bus. Data is available within one access time (t_{AA}) from the last address change. The chip select control (CS) functions as an output-disable. The CDP1832 is pin-compatible with the 512x8-bit erasable 2704 memory and can be directly inserted into a 2704 socket without any PC board changes.

In addition to the CDP1831 and CDP1832, the following 8192-bit mask-programmable CMOS ROMs, organized as 1024 8-bit words, are in development: the CDP1833, with address latch; and the CDP1834, without address latch.

The 1800 family offers, in addition to RAM and ROM circuits, memory support circuits to simplify the design of memory systems.

The CDP1858 address latch/decoder provides all the control required to interface the CDP1802 microprocessor with up to 4 kilobytes of RAM built from 256x4 RAM circuits.

The CDP1859 address latch/decoder is similar in function to the CDP1858 except it is designed to interface the CDP1802 with up to 4 kilobytes of RAM configured from 1024x1 RAM circuits.

The CDP1852 byte I/O is directly compatible with the CDP1802 for use as an 8-bit address latch.

The CDP1856 memory-bus buffer/separator can function as a bus buffer for larger memory systems and as a bus separator for split-bus memory devices. It is directly compatible with the CDP1802.

I/O circuits

A number of general I/O support circuits are available and more that extend the CDP1802 I/O structure are in develop-

ment. However, for a minimal system, the CDP1802 accommodates serial and parallel data transfer directly. In addition, the CDP1802 is compatible with the industry-standard CD4000 series COS/MOS family. The result is the ability to design any conceivable I/O interface with compatible devices. An outline of the RCA 1800 series of I/O circuits follows.

The CDP1852 is an 8-bit input/output port that simplifies parallel data transfers between the CDP1802 and external devices.

The CDP1852 (Fig. 7), when programmed as an input port (mode=0) or an output port (mode=1), will interface directly with the CDP1802 microprocessor without additional components. When used as an input port, data is strobed into the 8-bit register by a high (1) level on the clock line. The high-to-low transition of the clock latches the data in the register and sets the SR service-request output to 0. The three-state output drivers are enabled by CS1·CS2=1, and the high-to-low transition of CS1·CS2 resets SR=1.

When the CDP1852 is used as an output port, data is strobed into the 8-bit register when CS1·CS·CLOCK=1. The data is available at the outputs at all times because the three-state output drivers are always enabled when the mode input=1. The service request pulse is generated at the termination of CS1·CS2=1 and is present (high level) until the next high-to-low clock transition.

A CLEAR control is provided for resetting the port's register and SR (input mode) or SR (output mode) signal. It is important to note that the polarities of service request (SR/SR) and chip-select 1 (CS1/CS1) change when the polarity of the mode input signal is changed.

The CDP1857 is an I/O bus buffer and bus separator.

It can be used for buffering the data bus, separating the bus data, and as an input or output port where data-latching is not required. The CDP1857 is a silicon-gate CMOS device in a 16-lead package.

In addition to the above devices the following circuits are in development for later availability:

CDP1851 Programmable I/O
CDP1853 N-bit decoder
CDP1854 UART (Universal Asynchronous Receiver-Transmitter)

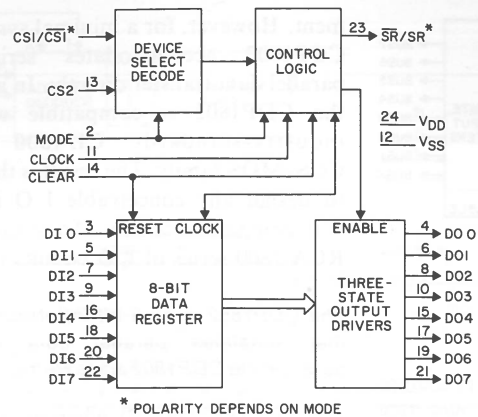


Fig. 7
Input/output port (CDP1852) performs parallel data transfers between the CDP1802 and external device directly.

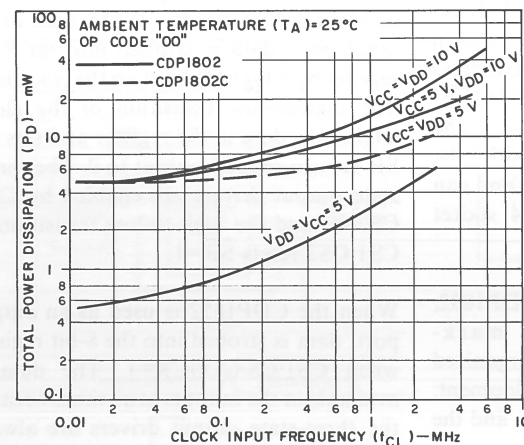


Fig. 9
Power dissipation for CDP1802 varies with input frequency and temperature.

Putting a system together

Designing a microcomputer based on the CDP1802 is a straightforward procedure. Generally, any microcomputer consists of a CPU, a variable amount of memory consisting of RAM and ROM, and an I/O interface. The design procedure consists of determining general system requirements or constraints, sizing and allocating RAM and ROM, including address space, and I/O design. Once the general microcomputer design has been mastered, the designer can focus more and more attention on the I/O, including the interaction of I/O hardware and software to minimize cost.

The first step in designing a microcomputer is to determine general requirements.

Specifically, operating speed, supply voltage, operating temperature range, and power dissipation must be established.

For a system requiring maximum performance, operating speed must first be established. Operating speed (or clock frequency)

will then fix the instruction time. For the CDP1802, all instructions require 16 clock periods, except long branches and long skips, which require 24 periods. Instruction time determines processor throughput and eventually I/O response time and resolution.

The CDP1802 maximum clock frequency is a function of supply voltage. Therefore, once the clock rate is established, the minimum operating voltage can be determined from Fig. 8. For systems that are not throughput limited, Fig. 8 can also determine a convenient clock rate and minimum supply voltage.

Next, the operating temperature range must be established. The processor clock rate is temperature-dependent and must be derated by 0.35%/°C. This dependency must be factored into the overall clock-rate/supply-voltage requirement. Fig. 9 shows power dissipation as a function of input frequency at 25°C. To account for temperature variations, a convenient approximation is to add 20 μ W/°C. This

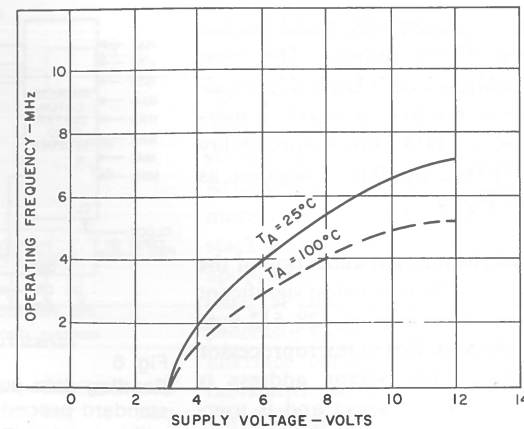


Fig. 8
Maximum clock frequency of CDP1802 is a function of supply voltage and temperature.

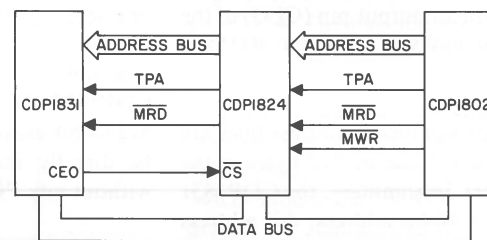


Fig. 10
Minimal COSMAC microcomputer system has only three chips.

factor is the increased quiescent dissipation, which is temperature dependent.

This example illustrates how to go about microcomputer design.

Design a microcomputer meeting the following system requirements:

- 1) Add two eight-bit bytes in 8 μ s
- 2) Operation from 25 to 100°C
- 3) Supply voltage must not exceed 7 V

Because the processor has an ADD instruction that adds M(R(X)) to D, one instruction will satisfy the 8- μ s add requirement. Clock frequency is $1/(8 \times 10^{-6} \div 16) = 1/500 \times 10^9 = 2$ MHz. Therefore, the microprocessor must operate at 2 MHz at 100°C. To determine the minimum supply voltage, derate the 25°C curve in Fig. 8 by $(100^\circ\text{C} - 25^\circ\text{C}) (0.35\%/^\circ\text{C}) = 26\%$. At 2 MHz, the supply voltage from the derated curve is 4.8 V. As a result, the CPU voltage can be selected between 4.8 and 7 V, depending on the desired margin and power limitations. Assuming a 5-V supply is chosen, the power dissipation can be read from Fig. 9 as 5 mW for the CDP1802D and 9 mW for the CDP1802CD, both at 25°C. For 100°C, these values are increased by $(100^\circ\text{C} - 25^\circ\text{C}) (20 \mu\text{W}/^\circ\text{C})$, or 1.5 mW.

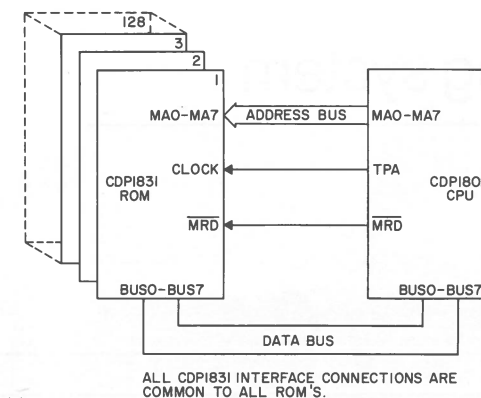


Fig. 11
ROM expands easily from 512 bytes to 65,536 without address decoding.

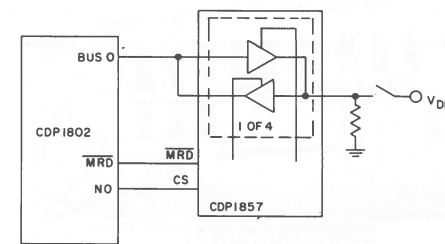


Fig. 13
Programmed I/O data transfer is possible with the CDP1802. Three ports can be selected directly, 14 with the addition of an N-bit decoder, and the number becomes virtually limitless with a two-level I/O convention. Diagrams show simple binary input port (left) and decoded parallel output port.

The designer should try to minimize variable-storage RAM and maximize invariable-storage ROM for the most cost-effective design.

The design of COSMAC microcomputers starts with the minimal system shown in Fig. 10. The memory system consists of 32 bytes of RAM (CDP1824) and 512 bytes of ROM (CDP1831). These devices are directly compatible with the CDP1802 and require no additional devices to interface with the processor. The address space for the system is determined by the user when he programs the CDP1831. The seven-bit user-specified sector address (seven most significant bits of the 16-bit address multiplexed to the CDP1831 8 bits at a time) locates the CDP1831 in one of 128 512-byte memory sectors. When the proper sector address is recognized by the CDP1831, the CEO output goes high and disables the RAM. The RAM, which is enabled when the ROM is not selected, occupies all 256-byte memory pages except the two occupied by the ROM.

Building onto the minimum system is also straightforward. Fig. 11 shows how the ROM can be expanded from 512 bytes to 65,536. No address decoding is required and the system is directly compatible with the CDP1802. RAM can be added as

before and enabled with the "OR-ed" CEO outputs of the CDP1831 ROMs.

The CDP1824, CDP1822, and CDP1831 memories are compatible with the CDP1802 at maximum processor speed. However, for large memory systems, the memory-system access time may not be speed-compatible with the CDP1802. To assist in determining memory-system speed requirements for the CDP1802, Fig. 12 shows the required memory-system access time for a specific instruction time or clock frequency.

The I/O section of a microcomputer is the least structured area of the system.

However, the I/O interface for the CDP1802 has been designed for flexibility, ease of use, and low cost. The simplest way to bring data into the CPU is via the four external flags, EF1-EF4. These inputs can accommodate independent serial data links, a terminal for example, or can be used together to input 4-bit data characters. The actual transfer of data is under program control. Branch instructions test the flag inputs and branch on the appropriate condition. Loops can be constructed to continually test the flag lines and manipulate the data as required.

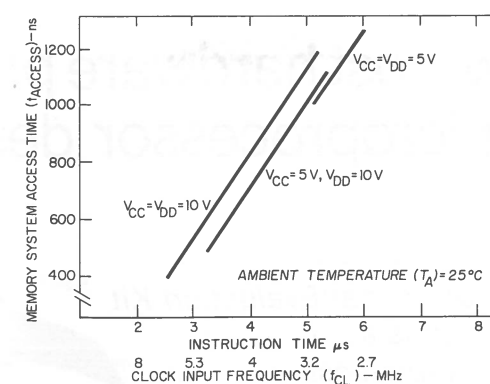
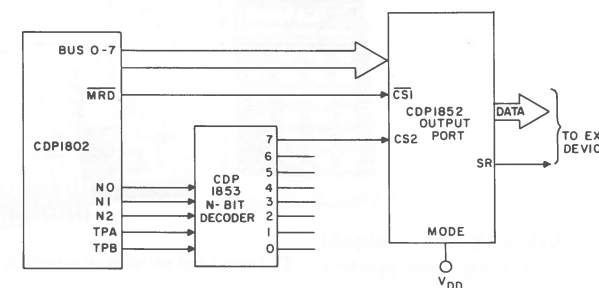


Fig. 12
Memory-system access time must be compatible with the microprocessor. Required access times for specific instruction times and clock frequencies are shown here.



Complementing the four input flags is a single-bit output port, Q, which can be set, reset, and tested by branch instructions. Under software control, Q can be used as a pulse-width-modulated, variable-frequency, or serial-data output port.

The CDP1802 also offers a programmed I/O data-transfer mechanism.

The processor I/O instructions are of the 6N format, where N is a variable input or output device-select code. The three least-significant bits of the binary N-code are available at dedicated processor pins for device selection. The CDP1802 can therefore select up to three I/O ports without additional components. This number increases to 14 by using the CDP1853 N-bit decoder; by adopting a two-level I/O convention, the number of external devices becomes virtually limitless. Examples are shown in Fig. 13. It should be noted that the N lines can be used as outputs in some systems.

The I/O design can become increasingly more sophisticated by using the built-in DMA and interrupt mechanisms in conjunction with the previously described I/O interfaces. State-code outputs are provided to indicate when the processor is in the DMA or interrupt states.

A low-cost hardware prototyping system for microprocessor design

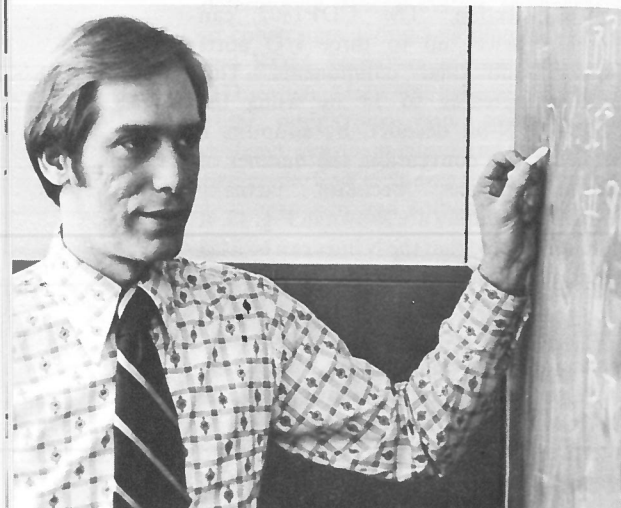
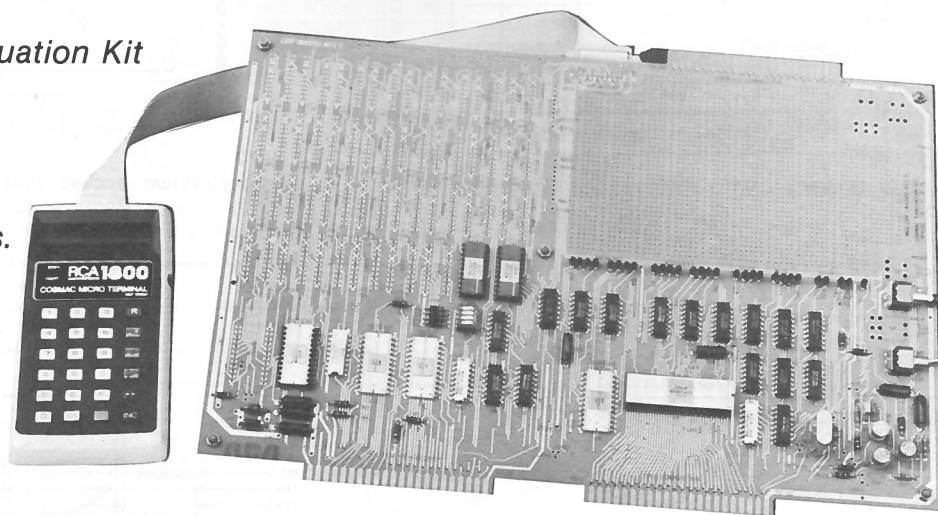
The Microterminal/Evaluation Kit combination is a convenient low-cost design tool for CDP1802-based microcomputer systems.

D. Block

Dennis Block became one of the early members of the RCA COS/MOS activity with his transfer to the Solid State Division in 1968. His work there included design of both standard and custom circuits and applications of the CD4000 family. Presently he is working in the microprocessor area with responsibility for applications assistance on standard products, support development and documentation, and related marketing activities.

Contact him at:
Microprocessor Applications Engineering
Solid State Division
Somerville, N.J.
Ext. 6046

Final manuscript received December 18, 1976.



A hardware prototyping system is a "must" in microprocessor system development. It is the equivalent of the breadboard in traditional logic design and has the same purpose—testing and verifying a design before it is committed to production.

RCA is supporting the CDP1802 microprocessor with a wide variety of prototyping systems for both hardware and software development. This paper describes a new low-cost microprocessor system and portable terminal designed for users of RCA 1800-series components or anyone with limited resources to invest. The system is also suitable for the hobbyist.

The system consists of the COSMAC Evaluation Kit, which is a microcomputer system based on the CDP1802 microprocessor, and the Microterminal for I/O communications. A user-supplied 5-V, 1-A power supply turns this system into a complete microcomputer with substantial capability and considerable flexibility. Although each of the two units can be used separately, together they open the door to expanded areas of microcomputer applications.

The COSMAC Evaluation Kit

An effective hardware-prototyping system should meet several criteria. First, it should be readily expandable, both in memory and in I/O capability. It should also provide a

means of program entry and verification, plus debugging features such as single-step and CPU register readout.

The Evaluation Kit meets these requirements. A kit of components for building a microcomputer based on the CDP1802 COSMAC microprocessor, it contains a PC board, CPU, byte input and output ports, terminal interfacing, a ROM-based utility program, and a RAM for user program storage. The RAM memory supplied is 256 bytes, but the PC board is prewired with memory addresses and chip-select signals so that up to 4 kilobytes of RAM can be accommodated by adding more CDP1822 memories. A full 16 bits of memory addressing is provided. Also, a battery backup option is made feasible by the use of CMOS memories.

A 6" x 4" area of the board is free for user-added I/O devices. ICs of various pin counts can be inserted into prepared positions in this area and jumpered to an uncommitted 44-pin connector on the board. Two other 44-pin connectors provide access to all CPU and system signals.

Several options exist for terminal interfacing. Both a conventional 20-mA current loop and an EIA RS232 interface at 110 to 1200 baud are available, which, along with a ROM utility program, provide

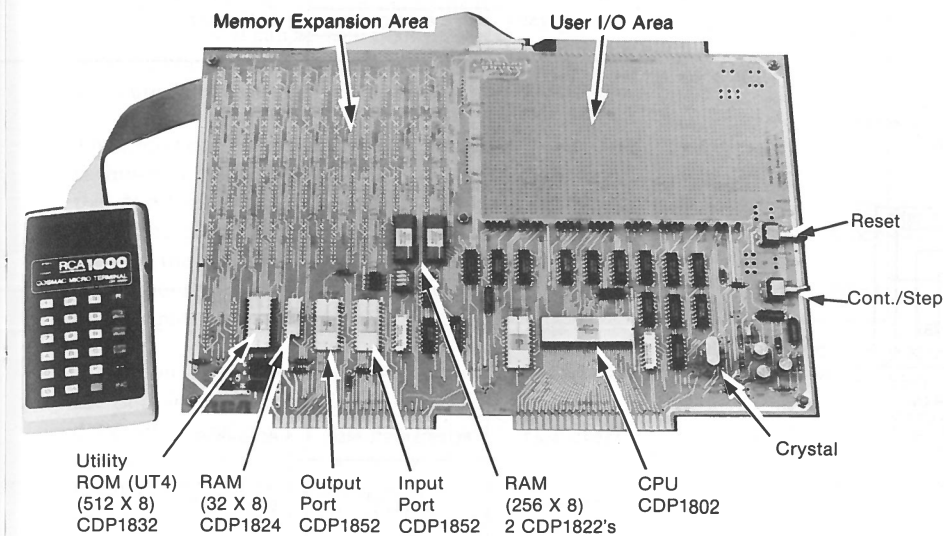


Fig. 1
Assembled Evaluation Kit PC board shows the large amount of space available for user I/O options and memory expansion.

automatic interfacing to ASCII serial terminals. A third option is the RCA Microterminal, which was designed to perform commonly required functions such as a memory inspection and modification, as well as numerous other functions to be described later.

Fig. 1 shows an assembled Evaluation Kit board; more detailed descriptions of the CDP18S020 system components follow.

The heart of the microcomputer system is the COSMAC CDP1802 microprocessor.

The CDP1802 is a CMOS LSI microprocessor that offers the advantages of CMOS technology, including full military temperature range, wide 3- to 12-V operating range, low power consumption, excellent noise immunity and fully static operation. It also has the unique COSMAC architecture, an efficient structure of general-purpose and control registers designed for a compact instruction format and flexible I/O interface.

The Evaluation Kit has been designed to assist the user in understanding the basic CDP1802 architecture as well as in applying the I/O control and data signals provided by the microprocessor. One feature of the Evaluation Kit is direct access to all CPU terminals; this means that the user can monitor CPU operation or

provide an input for a CDP1802 remote emulation function.

A detailed description of the CPU architecture and instruction set is provided in the *User Manual for the RCA CDP1802 COSMAC Microprocessor*, MPM-201A, supplied with the Evaluation Kit. Electrical specifications are given in the CDP1802 data sheet.

The CPU clock is generated by an on-chip crystal-controlled oscillator.

There are eight clock pulses per machine cycle and two or three machine cycles per instruction. A 2-MHz crystal is supplied; however, the user can design systems with much faster instruction times. For example, a 6.4-MHz clock frequency at 10 V results in instruction times of 2.5 and 3.75 μ s, and other instruction times can be obtained as a function of supply voltage and crystal frequency. There is no minimum frequency requirement because the CDP1802 is an entirely static device. Clock frequency changes can be made by replacing the provided crystal with one of the desired frequency. Alternatively, an external clock can be brought in via the CPU connector.

The control section initiates program execution and facilitates hardware and software debugging.

It has four controls with associated logic:

RESET; RUN U (run utility); RUN P (run program); and CONTINUOUS/STEP.

These four controls generate signals that are connected to the $\overline{\text{CLEAR}}$ and $\overline{\text{WAIT}}$ pins of the CDP1802. These two pins in turn control the four modes of operation for the microprocessor: LOAD, RESET, PAUSE, and RUN.

The Evaluation Kit control logic has been designed to permit operation of the system in two general modes: CONTINUOUS—normal run; and STEP—single cycle. By switching to the STEP mode and continuously pushing RUN P the user can sequence his program one machine cycle at a time and watch the system action on the LED display.

Discrete LED displays provide a visual indication of CPU operations so that program bugs can be spotted.

Displays are provided on the memory address bus, data bus, and CPU status signals. They can be used with the Microterminal to display the contents of 15 of the CPU's general-purpose registers, as discussed later. The LEDs go ON to indicate a logic 1 on the line being monitored and are OFF for a logic 0 or floating data-bus condition, because the data bus is provided with pull-down resistors.

The CDP1802 state code lines, monitored by LEDs, uniquely define the four microprocessor machine states: FETCH, EXECUTE, DMA and INTERRUPT. This information is valuable in debugging operations.

The Evaluation Kit is provided with an extensive, expandable memory system.

The memory consists of:

- 256-byte RAM prewired for expansion to 4096 bytes;
- 512-byte utility-program ROM;
- 32-byte utility-program RAM; and
- an optional 512-byte prewired ROM location.

The memory system interfaces with the CDP1802 microprocessor via address-latch, decode, and read/write control logic. Figs. 2 and 3 show the memory system in block-diagram and memory-map form.

The CDP1802 microprocessor itself is capable of directly addressing 64 kilobytes

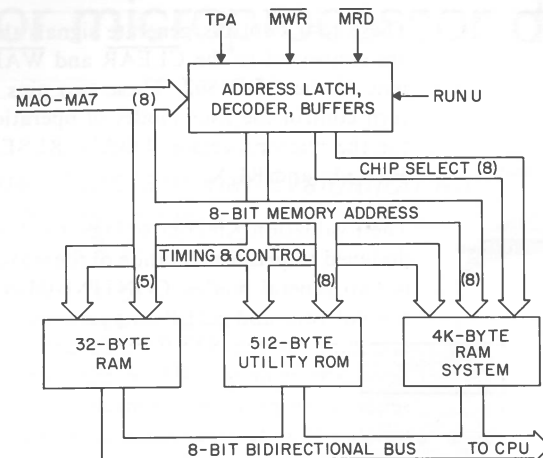


Fig. 2
Memory system for the Evaluation Kit is expandable to 4K of RAM and 512 bytes of prewired ROM.

of memory in any combination of RAM or ROM without restriction. The 16-bit address is multiplexed 8 bits at a time on the 8 memory address lines from the CDP1802. The CPU generates a timing pulse, TPA, during which the most-significant memory-address bits are present on the 8 address lines. TPA is used to latch the address bits required to form the full memory address. One-half clock period after the termination of TPA, the 8 least-significant address bits are multiplexed onto the address bus and remain valid for the remainder of the machine cycle.

The Evaluation Kit uses a CDP1852 I/O chip to latch the most significant byte with TPA. The remainder of the memory-system control logic is used to decode the 16-bit address for memory-chip selection.

The Evaluation Kit is provided with a 256-byte CMOS RAM consisting of two CDP1822 256×4 RAMs. The RAM system has been prewired for expansion to 4096 bytes by simple insertion of additional CDP1822s in designated locations. The RAM system is assigned the first 4096 locations of the 64-kilobyte system as indicated in the memory map of Fig. 3.

The Evaluation Kit also has a "write protected" RAM in 1-kilobyte segments. The memory write control (MWR) is connected to the RAM system through four DIP switches. When the switches are closed, the RAM system operates normally for both read and write operations. When a

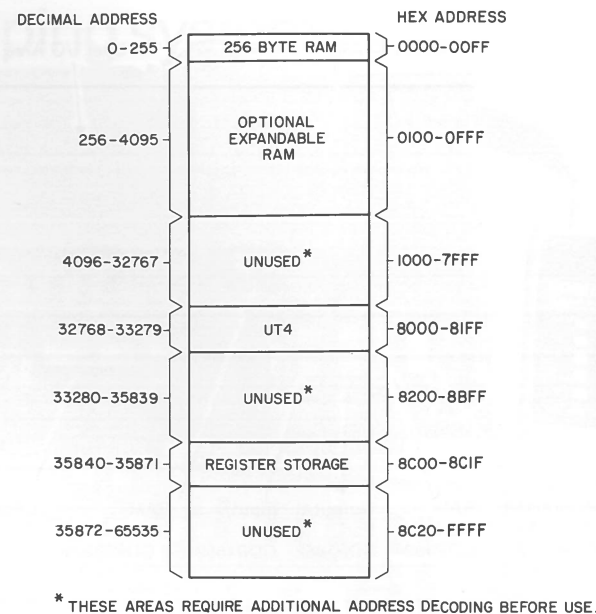
switch is open, MWR is inhibited from controlling the specified 1-kilobyte RAM segment and forces the RAM to a "read-only" mode.

The ROM system consists of a CDP1832 512-byte ROM factory-programmed with the utility program UT4. The CDP1832, a static CMOS device, is addressed by nine address lines and puts data on the bidirectional data bus within one access time of the last address change when selected. The Evaluation Kit design locates the utility program in the upper half of memory. UT4 allows the user, via teletypewriter or similar conventional terminal, to inspect and modify memory and to start program execution at a given location. It also handles terminal interfacing, including automatic speed adjustment. When started, the utility program will save 13-1/2 of the CPU's registers in its associated RAM, where they can be subsequently read out to determine CPU status just before UT4 was entered.

A CDP1824 32×8 CMOS RAM provides a register-save feature and scratchpad area for the Utility Program. It is selected by "ANDing" MA15 (8000) with a decoded 0C00 address state.

The I/O section has been designed for flexibility, accessibility, and easy experimentation.

The Input/Output section of the Evaluation Kit consists of an 8-bit parallel-output port, an 8-bit parallel-input port, and a 1-of-8 decoder for selecting I/O devices.



* THESE AREAS REQUIRE ADDITIONAL ADDRESS DECODING BEFORE USE.
Fig. 3
Memory map for Evaluation Kit. UT4 is a utility program in ROM.

The output port is designed to respond to CDP1802-programmed output-data transfers. The microprocessor I/O instructions are all of the 6N format, where N is a variable-input or -output device-select code. For output instructions, N is 1-7. The binary N code is available on CPU pins 17 through 19 for device selection.

MRD is used to indicate whether the operation is an input or output; MRD=0 for output instructions and MRD=1 for input instructions. Therefore, gating MRD with the N-lines uniquely defines device selection and direction.

Fig. 4 shows the circuit configuration for the output port. The three CDP1802 N lines are connected to the 1-of-8 N-bit decoder. Output "5" (pin 6) of the decoder is connected to the CDP1852 chip-select input CS2 and used to select the port. The CDP1802 MRD (memory read) output is connected to CS1 of the output port and used as the second select input. Output data transfers occur between memory and the output device. Data from the output port is available at the system connector, as indicated in Fig. 4. It should be noted that the N-bit decoder is not necessary for three or less I/O devices. For these cases, an N-line can be connected directly to the I/O-port chip-select input.

The input port, which operates in a manner similar to the output port, is designed to respond to CDP1802-programmed-input-data transfers. Input instructions are of the

format 6N, where N is 9-F. In analyzing the seven N codes, it can be seen that the three least significant N bits range from 1-7, as in the output instructions.

The circuit configuration for the input port is shown in Fig. 5. Output "6" (pin 7) from the N-bit decoder is connected to the CDP1852 chip-select input CS2 and used to select the port; MRD is connected to the CS1 input and used as a second (direction) select input.

Operation is as follows. Data is loaded into the port by an external device clock. The data inputs and clock signal are available on system connector pins as indicated. The negative CLOCK transition sets the service request flip-flop (SR) to 0. The SR signal can be connected to either EF3 or INTERRUPT of the CDP1802, allowing either an interrupt service routine or a software loop to respond to the service request for an input data transfer. The CPU, by executing a 6E instruction, will enable the input port onto the data bus and load memory and CPU register D with the latched data. At the completion of the 6E instruction execution, the SR signal will be reset to 1.

The TTY reader and printer interface electronics can be configured for either 20-mA current loop or EIA RS232C compatibility. All TTY connections are available at the system connector pins.

The user I/O area is a special feature of the Evaluation Kit.

This area is intended for constructing user-defined memory and I/O systems. It will accommodate any combination of 8-, 14-, 16-, 18-, 22-, 28-, or 40-pin dual-in-line packages. For example, the area can be used to assemble a system consisting of eighteen 16-pin packages or twelve 24-pin packages. To assist in assembling custom memory and I/O systems, convenient access to the CPU data bus (BUS0-7), memory address bus (MA0-7), and TPA, TPB, and MRD signals is provided on the left side of the user I/O area. The area also has an uncommitted user-I/O connector for interfacing with external systems, eighteen LED locations, and two switch locations.

The user I/O area plays an important part in customizing the Evaluation Kit to a specific application. Used together with the prewired memory and control functions, it enables the user to quickly construct and debug prototype microprocessor systems.

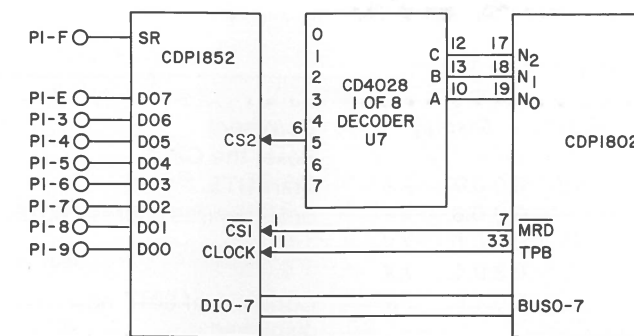


Fig. 4
Output-port circuit uses N-bit decoder where there are more than three I/O devices. Otherwise, N-line is connected directly to the I/O-port chip-select input.

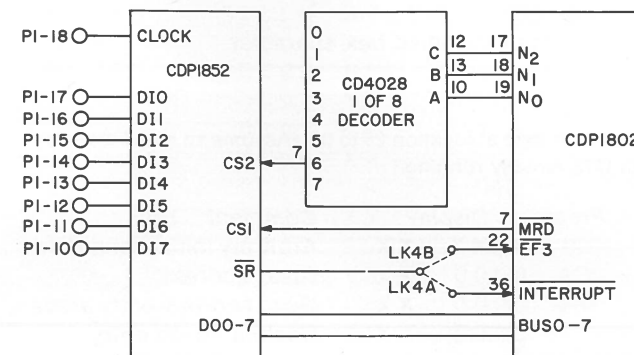


Fig. 5
Input port uses an external clock and service-request flip-flop to load in data.

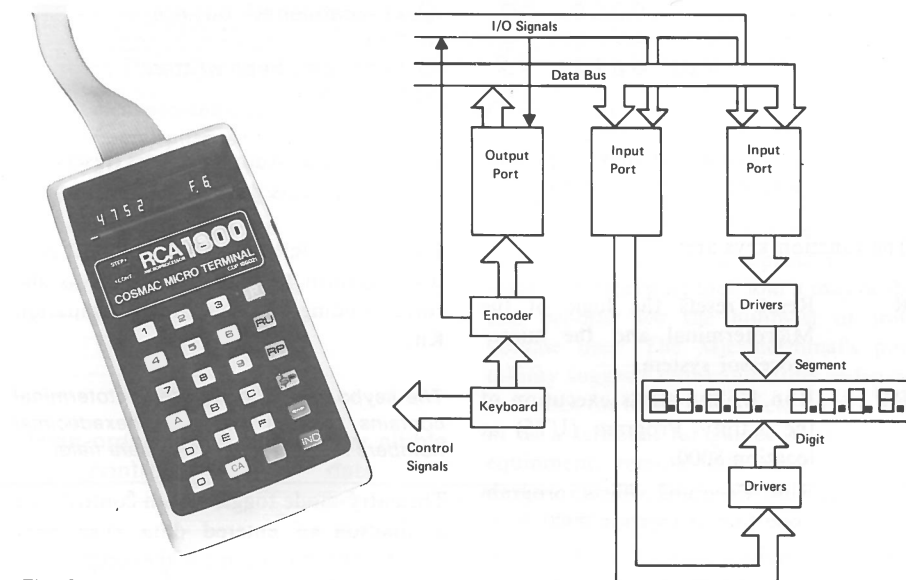


Fig. 6
Hand-held Microterminal is a low-cost alternative to the teletypewriter data terminal. It controls the microprocessor and displays registers and memory locations, making it an excellent debugging tool.

The Microterminal

The Microterminal (Fig. 6) is a small, portable, low-cost data terminal especially suitable for use with the COSMAC Evaluation Kit; it can also be used with a comparable user-designed system. The Microterminal consists of a keyboard and

display unit, with its connecting cable and mating connector, and a ROM containing the utility program to run the terminal and various subroutines that user programs can access. The Microterminal as shown is divided into three functional sections: control, display, and keyboard.

Table I
Read memory location 801F.

Enter	Press	Display	Comment
	R	—	Reset the CPU.
	RU	0.0.0.0. XX	Start UT5.
8		0.0.0.8. XX	Enter desired address, 801F.
0		0.0.8.0. XX	
1		0.8.0.1. XX	
F		8.0.1.F. F8	Contents of 801F now displayed.
To read contiguous addresses, press the INC button.			
	INC	8.0.2.0. 35	Contents of 8020 is 35.
	INC	8.0.2.1. A6	Contents of 8021 is A6.
		etc.	
X denotes an unspecified hex character.			

Table II
Change the data byte at location 20 to 00. (Assume an initial arbitrary state with UT5 already running.)

Enter	Press	Display	Comment
		XXXX X.X.	Arbitrary initial conditions.
	CA	0 0 0 0 X.X.	Clear address.
	↔	0.0.0.0. X X	Go to address-entry mode.
2		0.0.0.2. X X	Begin address entry.
0		0.0.2.0. X.X.	Address established.
	↔	0 0 2 0 X.X.	Go to data-entry mode.
0		0 0 2 0 X.0.	Begin data entry.
0		0 0 2 0 0.0.	Data established, but not written.
	INC	0 0 2 1 X.X.	Data has now been written to memory.

The control section can reset logic, run programs, and increment addresses.

The function keys are:

R	Reset: resets the logic of the Microterminal and the microprocessor system.
RU	Run Utility: starts execution of the Utility Program (UT5) at location 8000.
RP	Run Program: starts program execution at location 0000.
CONT	Continuous/Step: enables continuous or single-step operation of the microprocessor system.
↔	Entry Mode Control: toggles the mode between address-entry and data-entry.
INC	Increment Address: increments the address shown in the display. In the data-entry mode, it also has the data byte shown written into the address shown.
\$P	Start Addressed Program: starts program execution at the location shown in the address display.

CA Clear Address: Clears (resets) the address display to 0000.

The R, RU, RP, and CONT/STEP controls perform the same functions as the corresponding switches on the Evaluation Kit.

The keyboard section of the Microterminal contains 16 keys that enter hexadecimal numbers into the address or data field.

The entry-mode toggle switch controls the destination of entered data. The hexadecimal keys and some of the control keys are encoded and sent to the microprocessor system on a bidirectional data bus; logic scans and signals keyboard activity to the microprocessor. The actual scanning, debounce, and decoding algorithms are performed by software routines in the utility program.

The display section consists of an eight-digit seven-segment LED display, display drivers, and refresh logic.

The display shows a four-digit address field on the left and a two-digit data field on the

right; the two fields are separated by blank positions. In other subroutine-oriented display modes, all eight digits are available. Decimal points adjacent to either the address or data digits indicate whether the display is in the address-entry or data-entry mode.

A separate 5-V supply terminal is provided for the LEDs and their drivers. The rest of the logic is supplied from V_{CC}, which may range from 5 to 12 V.

The Microterminal interface is controlled by utility routines in its associated ROM.

These routines, collectively known as UT5, handle keyboard-scanning, debouncing and decoding, display-code conversion and multiplexing, standard modes of terminal operation, and display-function subroutines addressable by a user program. This ROM replaces the UT4 Utility ROM supplied with the Evaluation Kit for interfacing to teletypewriters or similar terminals.

There are three basic operations that can be performed with the Microterminal: memory read, memory write, and CPU register readout.

When the Microterminal is in the address-entry mode, the contents of memory at the location shown in the address field are displayed as a two-digit hexadecimal byte in the data field. When the utility program is started, it puts the terminal in the address-entry mode, denoted by lighted decimal points in the address field. It is then only necessary to enter the desired address by pressing the hexadecimal keys. Numbers are shifted in from right to left. See Table I.

When the Microterminal is in the data-entry mode, the byte in the data field is written to the address field location only when the INC button is pushed. The data-entry mode is signified by lighted decimal points in the data field. Hexadecimal numbers are entered in the data field from right to left. See Table II. To skip a location while entering data, simply increment past the unaltered locations.

Registers R1 through RF can be read out at the point where a user program is halted.

When this feature is used with a "planted" IDLE instruction, it provides the means for implementing an elementary breakpoint for debugging purposes. Assume a breakpoint is required at location XXXX of the user

program so that CPU registers and certain memory locations can be examined there. First an IDLE instruction would be written into location XXXX before starting the program. The user program will idle when it reaches that location, and registers can then be sequentially displayed in the address LEDs by repeatedly pressing RU while in the STEP mode.

Microterminal utility programs contain extra subroutines as a convenience for the user.

The Microterminal utility program, UT5, contains two main sections: a routine called KEYUT, which takes care of terminal interfacing, and a group of user-oriented subroutines.

KEYUT performs two main tasks: periodically refreshing the display; and scanning for keyboard inputs and executing the required actions.

KEYUT spends most of its time in a loop of refreshing the display and waiting for a key depression. When a key is pressed, the program decodes it and performs the required action. KEYUT waits for the key to be released (refreshing the display in the meantime) before returning to the main loop.

As a convenience to the user, the ENTRY subroutine initializes the CPU registers necessary for the COSMAC standard call and return technique. This technique allows multiple levels of subroutine nesting by using a stack in RAM. The call and return subroutines themselves are included on the ROM (UT5).

Thus, a long branch to ENTRY as the first three bytes of a user program saves writing the initialization code otherwise required to establish the standard call and return technique.

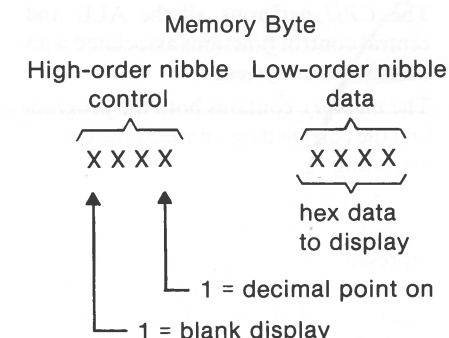
The COUNT subroutine is an independent program that displays memory sequentially starting from location 0000, incrementing the address approximately once per second. This subroutine provides an automatic read-back of a user program previously entered.

Two subroutines are provided to output data to the display via a user program. The subroutine REGDIS sends the contents of registers RA and RB to the display. Each register is displayed as a four-digit hexadecimal number with RA appearing at the left of the display.

Table III
Display registers RA and RB; increment RA.

Enter	Press	Display	Comment
	R		
	RU	0.0.0.0. X X	Start UT5.
	↔	0 0 0 0 X.X.	Go to data-entry mode.
C0		0 0 0 0 C.0.	Set up a long
	INC	0 0 0 1 X.X.	branch to ENTRY.
81		0 0 0 1 8.1	
	INC	0 0 0 2 X.X.	
08		0 0 0 2 0.8.	
	INC	0 0 0 3 X.X.	Go to location 0005,
	INC	0 0 0 4 X.X.	the return point
	INC	0 0 0 5 X.X.	from ENTRY.
D4		0 0 0 5 D.4.	Call to REGDIS.
	INC	0 0 0 6 X.X.	
81		0 0 0 6 8.1	
	INC	0 0 0 7 X.X.	
A6		0 0 0 7 A.6.	
	INC	0 0 0 8 X.X.	
1A		0 0 0 8 1.A.	Increment RA.
	INC	0 0 0 9 X.X.	
30		0 0 0 9 3.0.	Loop back to 0005.
	INC	0 0 0 A X.X.	
05		0 0 0 A 0.5.	
	INC	0 0 0 B X.X.	Program is now loaded. To start it running, press
R			
RP		XXXX XXXX	
		RA RB	

The other subroutine, called LEDD, is more general-purpose. It allows user control of all eight digits of display, plus their decimal points. LEDD reads out eight consecutive bytes of memory, starting at the location pointed to by RF, interpreting the bits at each location as a control and data character as follows:



Unused display positions are specified as blanks (8X) in the appropriate memory positions. LEDD leaves the data pointer RF at its initial value when it exits.

Table III demonstrates the use of REGDIS, one of the UT5 subroutines.

Conclusions

The Microterminal-Evaluation Kit combination offers a low-cost approach to microprocessor design. For designers of high-volume systems, it represents an attractive prototyping tool, and it may be the end product for the hobbyist or low-volume user. The Microterminal's portability suggests numerous other independent uses for it, such as a field service tool or as a terminal for games or small test equipment, remote data entry, teaching devices, etc. The Evaluation Kit can also be used independently or with a more sophisticated terminal such as a CRT to perform as a compact yet powerful single-board computer. But together these two components are a synergistic marriage of the best features of the microprocessor revolution.

Acknowledgments

The author wishes to acknowledge his debt to many, but particularly to A.W. Young, C.T. Wu, and R.J. Sedlak for their major contributions to the development of the systems and equipment described in this article.

Architectural trade-offs in designing microcomputer systems

P.M. Russo

Should it be ROM or PROM, one-level or two-level I/O, assembly language or a higher-level language, single-processor or multiprocessor, master-slave or master-master, ...?

Paul Russo joined RCA Laboratories in 1970; since then he has done research in computer architecture, program behavior, system performance evaluation, microprocessors, and microprocessor applications. He was involved, from the outset with the development of the COSMAC microprocessor and the original FRED system. Dr. Russo has taught several microprocessor courses and was the recipient of a Laboratories Outstanding Achievement Award in 1974.

Contact him at:
Systems Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 3231

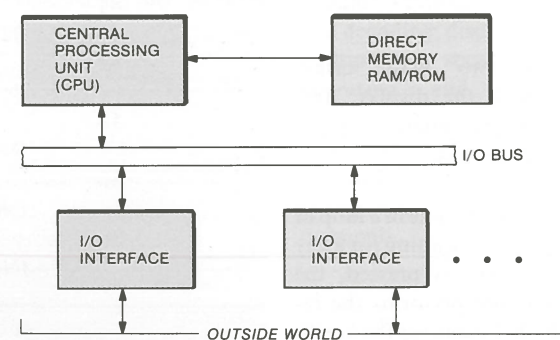


Fig. 1
Any computer system has these basic subsystems.

Microcomputer design resembles standard computer design in that it involves a number of choices and trade-offs. So, before one can address the general topic of microcomputer system design, the various subsystems associated with any computer system must be identified. Fig. 1 illustrates the three basic elements that make up any single-processor system:

The *CPU* performs all the ALU and central control functions associated with the computer system.

The *memory* contains both the program and the current data on which the CPU is operating.

The *I/O interfaces* are the key elements of any computer system, since they represent the only communication channels between the internal computer functions and the outside world (the I/O devices).

Any computer system, be it a large data processing center or an automobile ignition controller, is made up of the above three subsystems. However, differing applications dictate different I/O devices, different levels of performance, and different cost requirements. These, in turn,

result in vastly different CPU, memory and I/O interface requirements, different I/O architectural trade-offs, and possibly even entirely different device technologies.

Where the difficult trade-offs lie

This paper briefly examines the various architectural considerations that go into the design of microprocessor-based systems. The topics discussed include the basic system design steps; one-, two-, and multi-level I/O structures; the classical hardware/software trade-offs in interface design; and low- vs. high-volume product hardware/software cost trade-offs. Finally, there is a brief overview of multi-microprocessor structures, one of the new frontiers in microprocessor system architectures.

In general, selecting the appropriate device technology, CPU and memory organizations, and I/O devices is less difficult than making the subtle hardware/software trade-offs associated with the I/O architecture. Yet the I/O architecture is usually fundamental to the success of a system design and will dramatically affect cost and performance.

Microcomputer system design

The selection of any subsystem, be it CPU, memory, or I/O, cannot be made without examining the interrelationships that exist between all of them. However, since selecting the CPU and memory subsystems is relatively simpler than specifying the I/O architecture, the latter will be given greater emphasis throughout this article.

The properties of any CPU fall into three broad categories—functional architecture, chip architecture, and technology—whose importance depends on the application.

The CPU *functional architecture* is the key to its data handling and control capability. It includes features such as:

- Word size (e.g., 8 bits);
- Memory address range (e.g., 64K words);
- Instruction set; and
- I/O facility.

The CPU *chip architecture* defines the signals available or needed at the various chip pins. Hence, it defines both the amount of external logic the CPU needs to communicate with the outside world and the amount of logic and number of voltage levels required for basic CPU operation. The chip architecture will primarily influence cost, since it directly affects the ease of interfacing with various types of memory and I/O devices and defines the complexity of external clock-generation circuitry.

The CPU *technology* defines the chip density, power dissipation, drive capability, noise immunity and, usually, machine-cycle time. These factors affect chip cost (yield, chip size), system throughput (chip speed), and system cost (power supplies, external buffers to drive I/O devices, etc.).

Obviously, it is impossible to specify the characteristics of an optimal CPU *a priori*, since most desirable attributes, even within any one CPU category, are inherently conflicting. As an example, for a given number of pins, word size can be traded against memory addressing range, but both cannot increase simultaneously without affecting the amount of external support logic required and hence system cost. Excellent articles on microprocessor selection are available in the literature.^{1,2}

Memory selection, besides satisfying CPU speed requirements, is dictated by software needs (must software be changed often?) and the intended volume of the application.

In general, all microprocessor-based systems require some RAM and some type of read-only memory. The system designer has a host of memory types to choose from, each of which can usually be implemented in a variety of technologies. The following is but a partial list:

- Random-access read/write memory (RAM);
- Mask-programmed read-only memory (ROM);
- Programmable read-only memory (PROM);*
- Erasable programmable read-only memory (EPROM);**
- Electrically erasable programmable read-only memory (EEPROM).

Additionally, non-volatile memories (ones that will maintain data with power off) also exist (e.g., MNOS), but these are usually not suitable as main memories because of cost and speed considerations.

*PROM usually refers to memories that can be programmed only once (e.g., fusible-link PROMs).

**Usually the entire chip is first erased via uv light exposure so it can then be written onto. The write time is generally much longer than the read time.

In applications requiring flexibility and/or low volume, PROMs or EPROMs are often the best choice for read-only storage. For large-volume dedicated applications, where the program is frozen for relatively long time periods, ROMs usually offer the lowest-cost alternative.

The importance of the I/O architecture to the cost-effective implementation of microcomputer systems cannot be overstated.

I/O architecture is the key to success and involves the judicious trade-off between hardware and software for implementing system functions. The I/O interfaces are the only link with the outside world and hence are crucial from both human-engineering and system-capability points of view. I/O-related topics will be considered in depth below.

The COSMAC 1802 I/O structure, which is extremely elegant and powerful, will be used as the example vehicle throughout this paper. It is summarized in Appendix A and fully documented in Ref. 3.

System design steps—top-down approach

Fig. 2 shows the basic steps in the top-down approach to microcomputer system design. Note that the last three steps can be done concurrently. The following paragraphs examine each design step briefly.

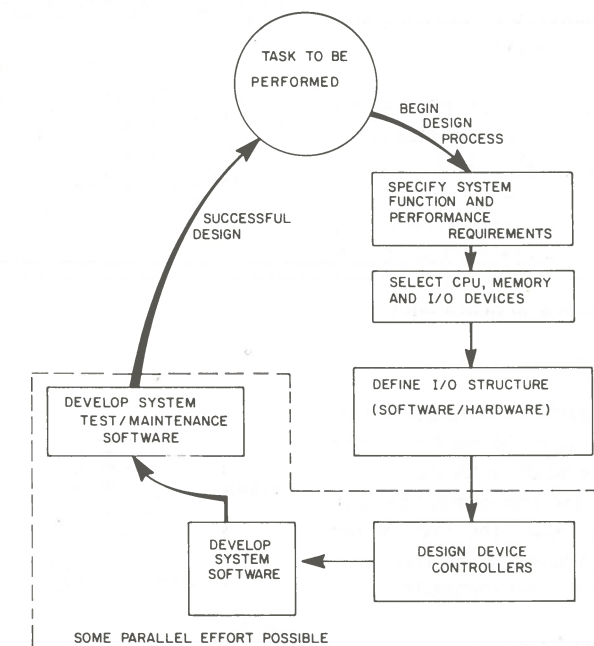
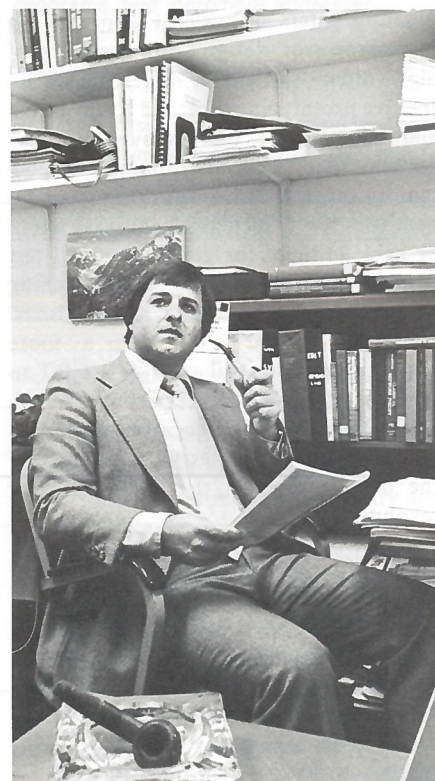


Fig. 2
Top-down approach to microprocessor system design is becoming more evident.

Final manuscript received October 25, 1976.



Specify system functions/performance.

The system designer must identify, at a high level, all the functions the system must perform. These may include monitoring and control, with various user interactions. Critical interrupt and other response times must be clearly identified along with the need for real-time operation. Finally, noise, power consumption, measurement techniques, diagnostics, and maintenance requirements must all be carefully spelled out. If effective simulation tools are available, this is the proper time to use them to their fullest.

Select CPU, memory, and I/O devices.

The system architect must then select the CPU, memory, and the I/O devices. This selection process must be made with full attention to noise, power, auto-restart, and environmental considerations. Typical I/O devices may include disc and tape units for mass storage, display devices, keyboards and switches for operator interaction, sensors and transducers for monitor/control functions, and backup power if un-interruptible operation is necessary.

Define I/O structure.

Having specified the I/O devices, the designer must then define how they will communicate with the computer system, both at the hardware and software levels. Additionally, each set of functions, for every given I/O device, must be partitioned into those suitable for software and hardware implementation.

At this stage of the design process, the system architect must indicate which device will use which CPU I/O facilities (interrupt, DMA channel, external flags, I/O instructions, etc.) and whether they will operate in a one-, two-, or multi-level I/O environment. The concept of multilevel I/O is covered in depth later.

Design device-interface hardware.

This is a relatively straightforward procedure. Care must be used to ensure that sufficient signal drive for device control is provided, and that compatible logic levels are used. Also, the logic family chosen must judiciously balance cost, power, speed, and noise immunity.

Develop system software.

Once the hardware is fully debugged, the software must be developed to control and

guide the hardware into performing its appointed task. Decisions must be made whether to program at the assembly level (more programming effort, but better memory utilization and higher throughput) or to use a higher level language (less programming effort, but less efficient use of memory and lower throughput).

A third alternative is to develop a custom interpreter to support a more "English-like" language. For example, a test language can be defined to enable relatively unskilled operators to develop complex test programs. A resident interpreter would then decode these "English-like" commands at run time.

Finally, a decision must be made regarding the software structure. Should it be modular and flexible (subroutine-oriented) for ready expansion, or should the code be

packed for maximum memory efficiency?

Perform system testing and provide diagnostics.

This last step is often given only lip service, yet it is crucial to the project's success. Not only must the designer ascertain that the system is properly packaged, that the power supplies are adequate, etc., but he must also specify test procedures so that repairs can be made efficiently. This may include the generation of good documentation and software diagnostics (test and maintenance programs) to help pin-point hardware failures or to ascertain proper system operation.

To summarize, the top-down approach to system design is far preferable with microcomputer systems. There is certainly a trend towards this philosophy.⁴

Choosing the I/O system

There are many ways in which microprocessors can communicate with the outside world. Not all microprocessors are capable of supporting all the following I/O techniques, but COSMAC, fortunately, can. The various ways that a CPU can interact with the I/O devices fall into two general classes—those totally under software control and those occurring asynchronously with normal processing.

Software-controlled I/O

There are three dominant categories of software-controlled I/O—programmed-mode I/O instructions, memory mapping, and serial input and output ports. Each of these techniques is briefly illustrated below. (See the Appendix for signal details.)

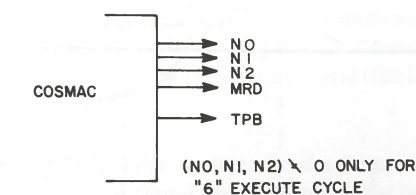


Fig. 3
Programmed-mode I/O uses MRD and three N bits to identify 61-67 instructions (memory to data bus) and 69-6F instructions (data bus to memory).

Programmed-mode I/O instructions are specific CPU operation codes that transfer information between peripheral devices and memory.

In COSMAC, a 61-67 instruction will move $M(R(X))$ * onto the data bus, whereas 69-6F will move the data bus contents (supplied by the peripheral device addressed) into $M(R(X))$. The CPU provides enough information to the peripheral device to allow it to ascertain when valid data is on the bus (memory to I/O) or when it must place its information on the bus (I/O to memory).

Referring to Fig. 3, we note that when MRD is a logical "1", the N bits identify a 61-67 instruction. Conversely, when MRD is at logical "0", the three N bits identify a

* $R(X)$ is a 16-bit general purpose register defined by the 4-bit X register. $M(R(X))$ is the contents of the memory location addressed by the contents of $R(X)$.

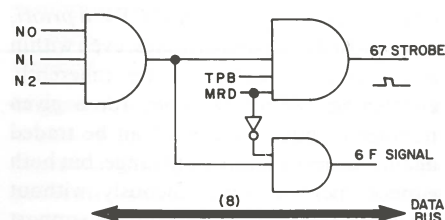


Fig. 4
Data is strobed out at TPB pulse in this example of programmed-mode I/O.

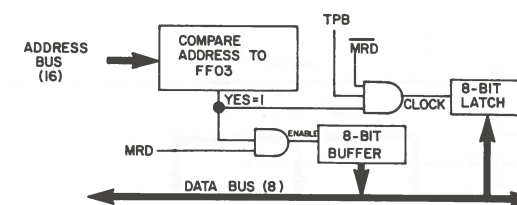


Fig. 5
Memory-mapped I/O assigns a memory address to each 8-bit I/O port (FF03 in this example). The I/O hardware then decodes the address and sends input or output data when required.

69-6F instruction. Since the N bits are zero unless an I/O execute cycle is taking place, they can be used to indicate when and which I/O instruction is taking place. Fig. 4 illustrates this situation. Note that the data on the bus is strobed out at TPB (output instruction), whereas input data is provided to the bus during the entire execute cycle.

The memory-mapped I/O concept is straightforward.

One views any 8-bit I/O port as a memory location. Memory addresses assigned to these I/O ports must not overlap the address range of the real memory system. The I/O hardware need only decode the desired address and input or output data as required. Any memory read or write instruction can be used to access these I/O ports. Fig. 5 illustrates the approach where the 16-bit address FF03 is assigned to this I/O port. When a memory read instruction is generated, MRD=1 and the buffer contents are held on the bus during the entire machine cycle. When a memory write is generated, TPB clocks the bus data into a latch.

Many microprocessors, including COSMAC, have serial input and output ports.

Serial input ports refer to physical pins on the CPU chip whose state can be sensed via software. COSMAC's four EF lines (see Appendix A) can be viewed as four serial-input ports. They, of course, may also be used in entirely different ways, as is shown below. When used as serial input ports, information is received by the CPU as a time sequence of logic states. Software routines convert these serial bit streams into bytes compatible with normal CPU operation.

Likewise, serial output ports are physical CPU pins whose level can be set via software. Software routines can convert 8-bit bytes into serial bit streams with the transmission rate and all timing fully under

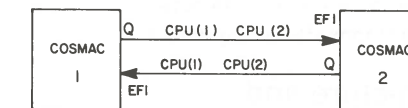


Fig. 6
Serial I/O ports are physical pins on the CPU chip whose level can be sensed by software. In COSMAC, for example, the four EF lines can represent input ports and the Q line can represent the output port.

software control. The Q output represents the serial output port in COSMAC.

The use of serial I/O ports usually results in a shift of the load from hardware to software. The hardware is usually very simple (and low cost) but the CPU is almost completely tied up during signal reception/transmission. Fig. 6 presents a trivial two-CPU system illustrating this method.

Asynchronous I/O

The two most common forms of asynchronous I/O are via interrupts and direct memory access (DMA) channels.

An interrupt facility may be viewed as a subroutine call mechanism under the control of external events.

When the interrupt line of a CPU is activated, the CPU branches to a specific location in memory, usually upon the completion of the current instruction execution. This location is the entry point of the interrupt service routine, which uses other forms of I/O to identify both the interrupting device and the specific cause for the interrupt. Additional I/O will then be used to service the interrupt, at the completion of which normal processing resumes where it left off.

A DMA channel provides a quasi-transparent path via the CPU between the memory and I/O subsystems.

Full control of the DMA channel is maintained by the specific I/O device having access to it. When a DMA input or output request is generated by the I/O device, the CPU will complete execution of the current instruction and then "hold its breath" for one machine cycle. During this time a byte can flow between memory and I/O devices and various counters and pointers are automatically updated to prepare the DMA channel for the next service request. Additional general information on DMA and interrupt facilities is available in Ref. 4.

COSMAC's interrupt facility, explained fully in Ref. 3, consists of a single interrupt line. When this line is activated, control automatically transfers to a subroutine pointed at by R(1), the new program counter. Hardware or software techniques can now be used to ascertain the interrupting device and the cause of the interrupt. Where speed is of the essence, hardware techniques can be used to implement both vectored and priority interrupt facilities. A vectored interrupt facility provides the ability to branch to different service routines for different interrupt causes. A priority interrupt facility may be required, in case of simultaneous interrupts, if certain devices must be serviced before others (because they might have higher data rates, for example). Software techniques for the above are usually based on polling the devices via testing appropriate EFs (see Refs. 5 and 6). Ref. 7 presents an implementation of a hardware interrupt priority-resolution circuit.

COSMAC's DMA facility, detailed in Ref. 3, provides the bulk of what is required to implement a full DMA channel. When the DMA-IN or DMA-OUT signals (see Appendix A) are activated by an I/O device, the CPU will complete the execution of the current instruction, then undergo a cycle steal (or DMA cycle), during which a byte will flow between memory and the I/O device via the data bus. The memory location is defined by R(0), which is automatically incremented during each DMA cycle. The I/O device, however, must count the number of bytes transferred and terminate the activity when the desired number of byte transfers has occurred. This counting capability must be implemented in hardware external to the CPU, and for that reason, the COSMAC DMA facility is not a full DMA channel.

The usefulness of COSMAC's DMA capability has been illustrated in two radically different applications—a home/school computer for tv display

refresh⁸ and a data-communications system for moving blocks of data between memory and a floppy disc subsystem.^{5,6}

I/O architecture and multilevel I/O structures

The design of the microcomputer's I/O architecture is one of the most critical steps in the entire system design process, as it will drastically affect both cost and performance. Most I/O subsystems will use a mixture of some or all of the techniques discussed in the previous section. Additionally, multilevel I/O structures often greatly magnify the system's I/O capability at relatively low incremental hardware and software cost.

Every CPU offers a different I/O facility, so the cost and performance trade-offs associated with any I/O technique will be unique to that processor. Hence, we will restrict our discussion to the COSMAC CPU.

With COSMAC, there is usually little need to go with a memory-mapped I/O scheme, since the hardware cost associated with using "6" instructions is minimal. If the system requires more than 7 input or 7 output ports, multilevel I/O structures can yield any desired number of I/O ports. As always, a few exceptions exist. In certain cases, such as where there is a real-time clock and the CPU may want to read the date and time of day stored in several sequential memory locations, a memory-mapped approach may prove effective (see Fig. 7). In this example, the CPU can obtain the date or time simply by reading the contents of memory location FFF0-FFF2 as required—no other I/O is required since the timekeeping is accomplished via external logic. When using "6" instructions for I/O with devices, the EFs may prove very useful in establishing system status. For example, they can let the CPU know that data is indeed ready to be read in. A simple system making full use of COSMAC's software-controlled I/O structure is the now-famous Microtutor.⁹

As system complexity grows, the need for more and more I/O instructions and EFs becomes desirable. This need can be satisfied by the use of two- and multilevel I/O structures.* Two-level I/O will be briefly described, with the note that the

*A single-level I/O system refers to one where all the I/O instructions, external flags, etc. are dedicated to unique functions for the entire system.

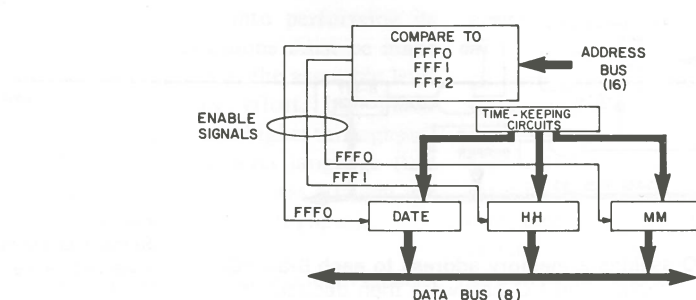


Fig. 7 **Memory-mapped** I/O is not normally needed with COSMAC, but this clock-reading circuit is one example of its use. Date, hours, and minutes are stored in memory locations FFF0, FFF1, and FFF2, and so are accessible by simply reading those locations.

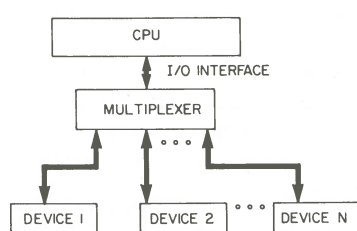


Fig. 8 **Two-level I/O** uses multiplexing to connect the software-controlled interface to up to 256 devices under CPU control.

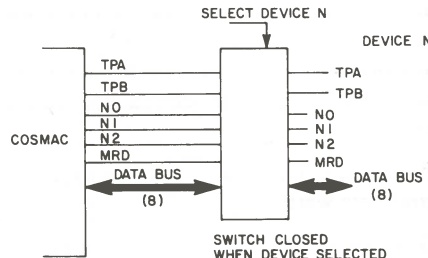


Fig. 9 **Multiplexing gates** only the software-controlled I/O lines.

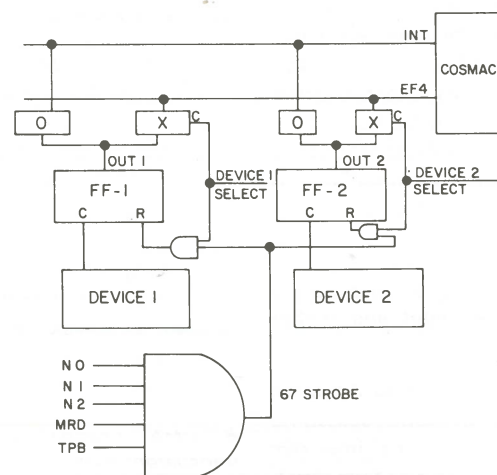


Fig. 10 **Two-level I/O system** determines the priority of multiple interrupts rapidly. Full explanation is in text.

same technique can be extended to more complex multilevel structures.

Two-level I/O uses multiplexing to increase the number of software-controlled devices.

The two-level I/O concept is best illustrated via Fig. 8. The multiplexer logic will connect the software-controlled I/O

interface to any of 256 devices under CPU control. In this way, the CPU can, at a given point in time, use its full I/O power in communicating with any one I/O device. In effect, after the CPU points at the device it wants to communicate with, only the selected device will be connected to the CPU I/O bus. Usually this multiplexing is achieved by giving up one output instruc-

tion (usually 61) to be the device SELECT instruction. When a SELECT instruction is executed, the contents of M(R(X)) enables one of 256 devices and attaches it to the I/O bus. All the other I/O devices will be cut off from the CPU until they, in turn, are selected.

Only the software-controlled I/O lines are so multiplexed—the interrupt and DMA lines are never used in this way, since this would negate their basic operational features. Thus, the signals that are gated via the SELECT switch are EF1-4, Q, TPA, TPB, N0-N2, MRD, and the data bus (see Fig. 9).

Once the two-level concept is implemented, then, except for the SELECT instruction, all the "6" instructions, EF1-4, and Q can have entirely unique meanings for each device. Hence, a 63 could imply "reset logic" for device 7 and "load byte count from data bus" for device 11. At any point in time, the system behaves as if the SELECTed device is the only I/O device in the system. More detailed information on implementing two-level I/O is available in Ref. 3.

When several interrupting devices exist in a two-level I/O system, the full flexibility in permitting a rapid priority determination of the interrupting device becomes apparent. This will be illustrated via a simple example. Consider Fig. 10, where both devices are permitted to interrupt the CPU. Device 1 has priority over device 2 when simultaneous interrupts occur. In both cases, the CPU will use the 67 instruction to acknowledge that interrupt servicing has taken place. The O blocks signify a wired-OR connection.* The X blocks signify transmission gates (when the control signal goes high, a path is established between the two other connections).

Let us investigate the operation of the circuit of Fig. 10. Suppose device 2 interrupts the CPU by setting flip-flop FF-2. Control will transfer to an interrupt routine where the CPU will first select device 1, the highest priority device, which will gate FF-1 onto the EF4 bus. The CPU will then test EF4—it will be inactive. The CPU now knows that device 1 is not the interrupting device. It will then select the next highest priority device—device 2. The selection of device 2 (61 with M(R(X)) = 02) will gate FF-2 onto the EF4 bus. The CPU will now

*In a wired-OR connection, when any input signal goes active, the bus signal goes active.

test EF4 and establish that device 2 is indeed the interrupting device. Upon servicing device 2, it will issue a 67 instruction, which will reset FF-2. The CPU will now resume processing where it left off upon interruption.

Suppose both devices had interrupted the CPU simultaneously; then, following the above sequence of events, the CPU would first select device 1 and service it, issue a 67 instruction to reset FF-1, select device 2 and service it, and finally issue a 67 to reset FF-2. The CPU would then resume normal processing.

Choice of language and system hardware

Both the selection of microprocessor language and system hardware are important to the cost-effectiveness of a microcomputer system.

The choice of language is difficult—and is really between assembly-level or higher-level (e.g., Intel's PL/M) code.

The advantages of assembly language over machine code are so many and so obvious that they will not be discussed here. The choice between assembly and higher-level languages is, however, non-trivial and depends strongly on the application. Where a large-volume product is being designed and the cost of each additional

Multi-microcomputer structures

The natural evolution of microprocessor-based system architectures brings about distributed processing, i.e., multi-microcomputer systems. In distributed intelligence systems, intelligent subsystems that are dedicated to specific tasks communicate in an optimal fashion to improve system throughput, increase reliability, and add a new dimension of flexibility.

Multiprocessor vs. multicomputer

Considerable confusion exists in the literature, since the terms "multiprocessor system" and "multicomputer system" are often used synonymously.

In general, multiprocessor systems are those that operate on a single input stream or workload. A single integrated operating

byte of memory gets multiplied by the number of systems sold, assembly language will usually be preferable because it gives the programmer machine-level control. The extra engineering required to generate software generation can be written off over many units. On the other hand, for a quick demonstration or a low-volume product, higher-level languages may well be preferable because of their advantages of documentation and rapid code generation.

There are also other considerations. For example, a well-written assembly-level program will usually execute faster than its higher-level counterpart. In general, about all that can be said is that assembly code may be faster, requires less memory, is more expensive to develop, and more difficult to document and update. Ref. 12 provides an excellent analysis of this problem.

The selection of a product hardware package is also not easy.

For large-volume applications, a custom package (and even custom I/O chips) may be optimal. For very low volume, on the other hand, it may be desirable to build the product around the CPU manufacturer's prototyping system, e.g., the COSMAC Development System. Again, the engineering and product costs associated with a new hardware design must be weighed against the ease of implementation but high unit costs of prototyping systems.

system allocates hardware resources where needed. These systems are used primarily in situations where high reliability is a must. This is achieved by either assembling a fully redundant system or by allowing for rapid system reconfiguration.¹⁰ Multicomputer systems, on the other hand, operate on several input streams, and do not have an integrated operating system.

Viewed in another light, the main function of multicomputer systems is to separate or partition the various tasks to achieve improved system throughput. Examples in larger systems include the separating of "number crunching," done in the main CPU, from "I/O processing" performed in various I/O processors. In multi-microcomputer systems, the primary motivation is to separate tasks that are mostly independent, i.e., ones that require

relatively little intercommunication. This partitioning enables the various microcomputers to be far more responsive to their dedicated tasks. In fact, each microcomputer may be controlling a process requiring, for example, rapid responses to interrupts. It would be difficult, perhaps impossible, for a single microprocessor to respond rapidly to a large number of interrupts.

Multi-microcomputer structures

In multi-microcomputer systems, each microcomputer, ideally, performs a dedicated task. Very little data is interchanged between processors relative to the total system data flow; each microcomputer can operate relatively independently of the others. The software in each subsystem is optimally tailored to the processing task for which it has responsibility. Should any one CPU become overloaded, it cannot be unburdened by any other processor; however, this should not occur with proper system design.

Multi-microcomputer systems, often called Distributed Intelligence Microcomputer Systems (DIMS), offer many advantages to the performance of the total system. The system can be modular in nature, where each subsystem is similar in hardware but customized in software. This greatly reduces maintenance and spare-parts inventory problems. Should one subsystem fail, the remainder can usually keep functioning, giving the system some "fail soft" capability, and hence, improved reliability. In DIMS, each subsystem is generally I/O-oriented, and the total system throughput, with careful design, can approach the sum of the throughputs of the individual processing elements.

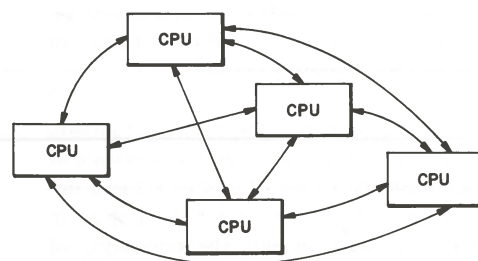


Fig. 11
Generalized network structure allows any CPU to communicate with any other CPU. Each CPU must have compatible interprocessor interfaces and I/O instructions.

Master-slave or master-master?

A myriad of possible DIMS structures exist. At one extreme lies the "master-slave" or "star" organization; at the other lies the generalized-network, or master-master, structure where every CPU has equal status. Intermediary structures and loop structures are also possible and may, in fact, be optimal for some particular applications.

Before discussing the applicability of any given structure in a microprocessor environment, it is worthwhile to examine the extreme structures. Fig. 11 illustrates a general network structure, in which any CPU can communicate with any other CPU. In this organization, all the CPUs must support compatible interprocessor interfaces and I/O instructions.

The "master-slave" organization, Fig. 12, offers many advantages to DIMS systems. In this organization, all inter-CPU communication must always go through the master. The principal advantage of this structure is that each slave CPU can have unique interprocessor I/O instructions, optimally tailored to its task. Hence, interface costs are reduced and extremely efficient use of I/O codes results.

Yet another possible structure is the ring structure illustrated in Fig. 13. Note, however, that if the information bus is also needed by the individual CPUs for their own processing, severe contention problems will ensue, with a resulting degradation in the performance of the overall system.

The master-slave organization appears to be the most suitable for the bulk of multi-microcomputer applications envisioned,

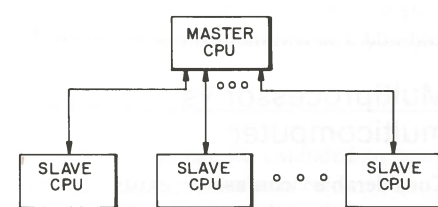


Fig. 12
Master-slave organization requires all inter-CPU communication to go through the master. In contrast to the generalized network of Fig. 11, this system allows specialized I/O for each CPU.

i.e., ones where a main CPU controls and supervises a multitude of intelligent (microprocessor-based) subsystems, each dedicated to a specific task.

Distributed intelligence always raises the question of shared main memory.

It is clear that, as the number of processing elements increases beyond two or three, severe contention problems will arise if each CPU must access common memory for a substantial fraction of its cycles. Thus, if a common memory is used, references to it by the individual CPUs must be minimized. Hence, some local RAM is highly desirable. In fact, insofar as multimicrocomputer systems are concerned, the primary use of a common memory would be to act as a message center, where each CPU can leave messages for other CPUs and pick up messages intended for it. Such an organization is illustrated in Fig. 14. Note that if there are N processing systems, N mailboxes are required, each one having $N-1$ compartments. Basically, if the i^{th} CPU wishes to leave a message to the j^{th} CPU, it simply stores the desired information in the appropriate compartment of the j^{th} CPU's mailbox. Any CPU can scan its mailbox to see if there are any messages for it.

The usefulness of shared memory in a master-slave organization is far more limited, since the number of paths is now equal to the number of slave processors, and hence is relatively small. A preferable approach in this case is to allow block transfers of data between the master and slave microcomputers via direct memory access (DMA) channels. The CPUs may communicate at lower data rates under program control, but data blocks will be

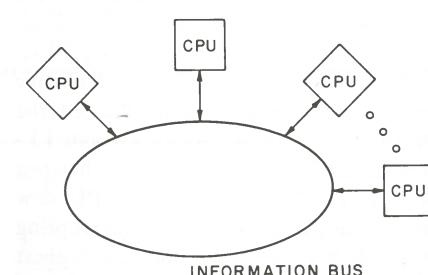


Fig. 13
Ring structure uses a single information bus to interconnect multiple CPUs.

transferred asynchronously with normal processing, thereby improving system throughput. This is the approach taken in the COSMAC interprocessor interface outlined below.

An interprocessor interface for COSMAC

The philosophy behind the current hardware design is as follows: a master-slave organization will be used with one master and an arbitrary (up to 256) number of slave processors. No common memory will be provided and each CPU will have sufficient RAM to handle its dedicated workload. The master CPU will have sufficient RAM to buffer all information that will be exchanged between it and all the slave processors. Interprocessor communication will be either via programmed-mode I/O (type "6" instructions) or via DMA for block transfers. The external flags will be used for handshaking and status information. Finally, the master processor can interrupt any slave, but no slave can interrupt the master.

The system organization is such that each slave CPU will have a dedicated interprocessor interface associated with it. These interfaces will be identical in hardware, with the exception of their device numbers. The master CPU is expected to operate in a two-level I/O environment with each slave CPU viewed as a peripheral with a unique device number. Slave CPUs may or may not use two-level I/O, but they, too, will view the master as a peripheral device.

Fig. 15 illustrates a three-CPU system consisting of a master and two slave CPUs, and hence two interprocessor interfaces.

Figure 16 presents a high-level view of the interface organization. Basically, two 8-bit buffers provide asynchronous communication between the master-CPU and slave-CPU data buses. Note that the two CPUs need not be synchronized, nor need they operate at identical clock rates. Besides the two 8-bit buffers, latches are provided for the external flags (EF1-4) of both CPU's, the master CPU's DMA requests, and the slave CPU's interrupt line. The control logic, which provides timing and clocking signals where needed, is not shown.

The basic hardware organization allows the exchange of information between the

master and slave CPUs under programmed-mode I/O via the two buffers and external flag decodes. The master is allowed to interrupt the slave, but not *vice versa*. High-speed block transfers between RAM(1) and RAM(2), or RAM(1) and the RAM of any slave CPU, run via the master's DMA channel, which must be

multiplexed among all the slave CPUs. This approach frees the slave-CPU DMA channels for use in conjunction with the subsystems to which they are dedicated (e.g., disc control). Each CPU, in effect, views the CPU it is communicating with as a peripheral device with which it can interact in an asynchronous fashion.

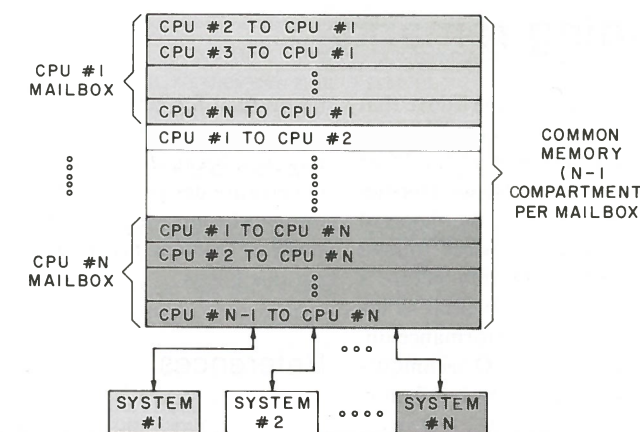


Fig. 14
Shared memory acts as a message center in a multiple-CPU organization. CPUs scan their "mailboxes" for messages from each other.

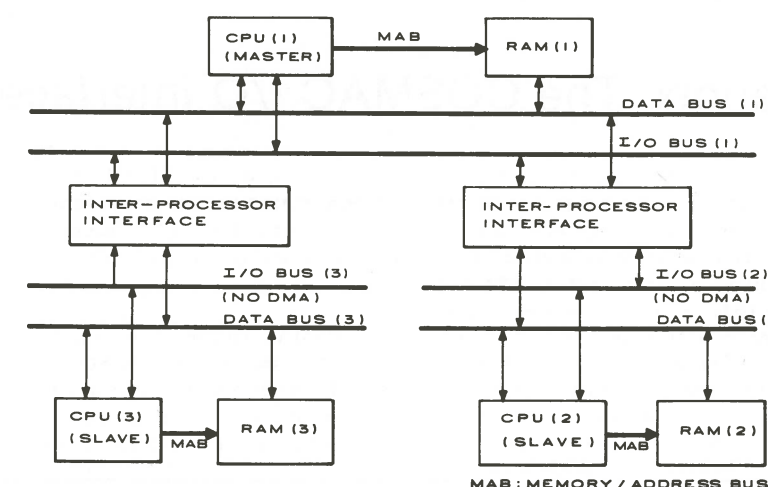


Fig. 15
This three-CPU system consists of master and two slaves; master views each slave as a peripheral with a unique device number.

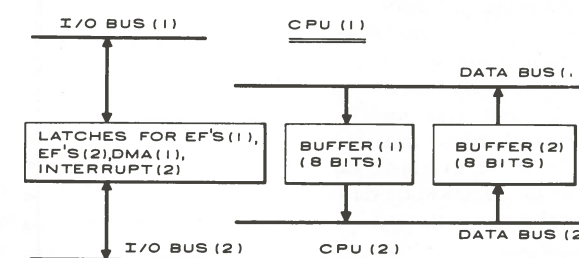


Fig. 16
High-level view of interface organization. Communication is asynchronous.

However, the master and slave CPU protocols are different.

The CPUs communicate via "6" instructions and EF assignments. These I/O instructions are used both for programmed-mode data transfer and to initiate DMA(I) block transfers between RAM(I) and any slave CPU's RAM. Ref. 11 details the above interface.

Concluding remarks

It should be apparent from the above that the design of microprocessor-based systems is part intuition, part art, and part science, but mostly common sense. Once an application is clearly defined, the choice of CPU, memory, and I/O devices follows readily. Designing the I/O architecture is more complex, but is fundamental to the system satisfying its cost/performance objectives. The right mix of I/O techniques usually depends strongly on the I/O facility of the selected CPU. Multilevel I/O structures, DMA, and interrupt facilities should be used effectively to satisfy desired cost and performance levels. Multi-microcomputer system architectures may

The advantages of the COSMAC inter-processor interface's organization are numerous. Foremost is the complete lack of contention, since each CPU can make full use of its cycles. Each slave CPU is free to use its DMA channel to communicate with the outside world. The master's DMA channel is time-multiplexed among the various slave CPUs under its control. The master can interrupt any slave, but not *vice versa*.

be optimal for certain applications. They should be approached with caution, however, because of the increased difficulty in software design due to the requirements for synchronization between the various software subsystems associated with the various CPUs.

References

1. Kaye, D.N.; "How to pick a microprocessor, a mini or anything in between," *Electronic Design*, Aug 2, 1975, pp. 26-30.
2. Weissberger, A.J.; "MOS/LSI microprocessor selection," *Electronic Design*, Jun 7, 1974, pp. 100-104.
3. "User Manual for the CDP1802 COSMAC Microprocessor," MPM-201A, RCA Solid State Division, Somerville, N.J.

The architecture is open-ended, since most of the bit assignments are user-specified via software. Hence, each interface can be customized in software while the basic hardware remains unchanged. Finally, the communication protocol is such that every processor can operate asynchronously, with a different clock rate. Thus, overall system synchronization is not required.

4. Raphael, H.; "Evaluating a microcomputer's input/output performance," *Electronics*, Aug 17, 1976, pp. 105-109.
5. Russo, P.M.; and Lippman, M.D.; "Case history: store and forward," *IEEE Spectrum*, Sep 1974, pp. 60-67.
6. Lippman, M.D. and Russo, P.M.; "A microprocessor controlled for international leased data channels," *Proc. of the ICC*, Minneapolis, (Jun 1974).
7. Marcantonio, A.R.; "Interrupt-priority resolution circuit for use with RCA COSMAC development systems," ICAN-6485, RCA Solid State Division, Somerville, N.J.
8. Weisbecker, J.A.; "A practical, low-cost, home/school microprocessor system," *IEEE Computer*, Aug 1974.
9. "COSMAC Microtutor Manual," MPM-109, RCA Solid State Division, Somerville, N.J.
10. Enslow, P.H., Jr., (editor); *Multiprocessors and Parallel Processing*, (John Wiley and Sons, Inc.; New York; 1974).
11. Russo, P.M.; "An interface for multi-microcomputer systems," *Proc. COMPCON 76*, Washington, D.C., (Sep 1976).
12. Gibbons, J.; "When to use higher-level languages in microcomputer-based systems," *Electronics*, Aug 7, 1975, pp. 107-111.

Appendix: The COSMAC I/O interface

The CDP1802 I/O interface is illustrated in Fig. A-1. It consists of an 8-bit data bus and 16 control lines. These 16 control lines break down into 4 lines used for synchronization and CPU state identification (timing pulses, TPA and TPB, state codes SC0, SC1), 4 lines used to identify type "6" I/O instructions, (3 "N" lines combined with memory read to decode 61-67 output, 69-6F input), 4 software-testable external flags (EF1-4), one software-controlled output level, one interrupt line, and two lines for controlling COSMAC's built-in direct memory access (DMA) channel.

From the above, we can identify four distinct ways in which COSMAC can communicate with the outside world via its I/O interface. Three of these provide both input and output capability—they are programmed-mode I/O (type "6" instructions), direct software testing or setting of levels (EF1-4, Q), and the DMA channel. The fourth way is via the CPU's interrupt capability, which can only alert the CPU that some external event has occurred. Of course, COSMAC can also communicate with the outside world via its memory interface, as discussed in the text.

When multilevel I/O structures (see text) are employed, only the control lines associated with programmed-mode I/O, EF1-4, and Q should be

gated with the device SELECT line. The interrupt and DMA channel control signals should never be gated with SELECT since they control activities occurring asynchronously with normal CPU processing.

The COSMAC microprocessor supports four basic CPU states, identified by SC0 and SC1. Each CPU state is active for one machine cycle (8 clock periods). The CPU states are INSTRUCTION FETCH (S0), INSTRUCTION EXECUTE (S1), DMA CYCLE (S2), and INTERRUPT CYCLE (S3).

The I/O devices need to identify the S1 cycle for programmed-mode I/O, the S2 cycle for data transfer via DMA, and the S3 cycle to acknowledge that the interrupt has taken.

The two timing pulses available are TPA and TPB. TPA occurs early in the machine cycle and is used primarily to latch the upper memory address byte. TPB occurs late in the machine cycle and is used primarily to clock valid data from the bus. Of course, both TPA and TPB can be used to generate strobes to reset or activate devices as needed.

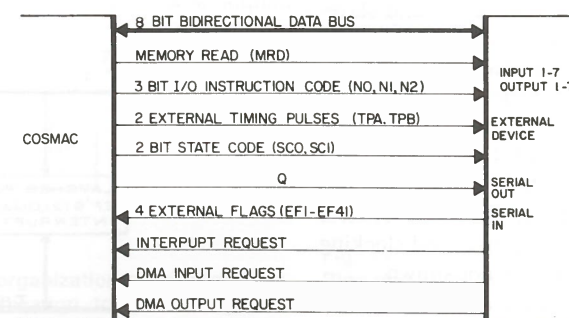


Fig. A-1
CDP1802 I/O interface consists of an 8-bit data bus and 16 control lines.

COSMAC resident software development aids

These self-sufficient aids make COSMAC microprocessor software development possible without the need for other computing facilities.

L.A. Solomon

The COSMAC Development System, CDP18S004, is a prototyping system for developing products based on the RCA 1800 microprocessor series. Referred to as the CDS, the system uses a CDP1802 microprocessor as a CPU and includes a complete family of software aids suited to a spectrum of cost/performance-conscious users. These self-sufficient software aids make complete software development possible without recourse to any other computing facilities.

The basic CDS is supplied with paper-tape and cassette versions of resident-editor and resident-assembler programs that allow the users to do program development and debugging on the COSMAC Development System itself. This self-contained system is an attractive alternative to the timesharing approach to software development.

For greater convenience in software design, the editor and assembler programs are also available on a floppy disk system CDP18S800. This system includes two disk drives, each capable of addressing up to 250 kilobytes of stored files on removable diskettes. The floppy-disk system has special CDS-compatible versions of the software aid programs and an interface card that plugs directly into the CDS.

Resident software

A resident software aid is a program that runs on a system (such as the CDS), is stored in or loaded into one of the system's memories, and performs some general function for the user. Such an aid may be classified in two ways. A permanently resident program is stored in ROM, occupying a fixed portion of the addressable memory space, and a temporarily resident program is loaded into some portion of the existing RAM space.

The permanently resident functions in CDS include utility and debug programs. The utility program loads memory with

temporarily resident programs and transfers control to them, so they can run; the debug program is for interactive diagnosis. The temporarily resident programs include the editor and assembler.

Assembler program

An assembler is a program that aids a user in writing his own application programs. Its function is to translate a program in assembly language into hexadecimal code, ready for machine operation. Although the user could write out and enter his program in machine language, an assembler offers several very important advantages, including:

- Mnemonic abbreviations rather than numerics for instruction names. This leads to fewer mistakes and makes "reading" the program easier.
- User-assigned mnemonics for the microprocessor's registers and locations in memory. This makes it possible to conveniently insert instructions or move parts of a program around; the assembler makes the necessary address changes automatically.
- Comments can be added to the program at will. This makes reading, debugging, or modifying the program easier, particularly if some time has passed since it was written.

To gain these advantages, the program is written in COSMAC assembly language, which is principally a symbolic representation of COSMAC instructions. The program may be typed in every time it is needed (not practical if it is long!), punched on paper tape, recorded on magnetic cassettes or on floppy disks, or stored in timesharing computer files. These approaches are successively more expensive, but also are increasingly easier and faster to work with. Whatever the program storage medium, the assembler translates the assembly-language program into hex-

adecimal code ready to run on a COSMAC system.

COSMAC resident assembler

The COSMAC resident assembler (CRA) program assembles COSMAC programs without the use of another computer. CRA runs directly on the COSMAC Development System itself in a stand-alone

Final manuscript received November 23, 1976.

Larry Solomon was the principal designer of the COSMAC resident assembler and its subsequent enhancements. He is presently a Leader in Software Development and Systems. He had five years of minicomputer system design for real-time data-collection and analysis systems before coming to RCA in 1974.

Contact him at:
Solid State Division
Somerville, N.J.
Ext. 6349



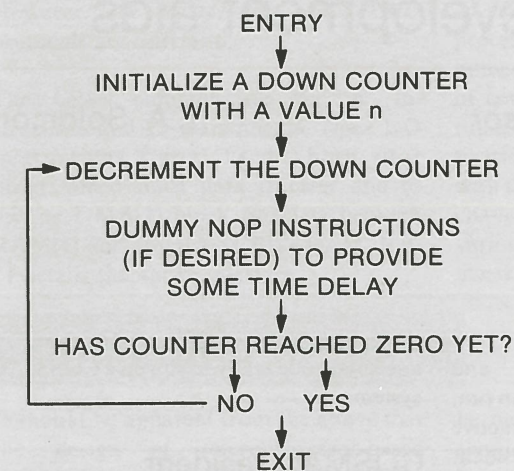


Fig. 1
"Time-out" program in flow-chart form.

F8FFB12191913A0300

Fig. 2
Machine-language version of the program of Fig. 1 is a number of steps away from the flowchart. Figs. 3 through 9 show how to get there.

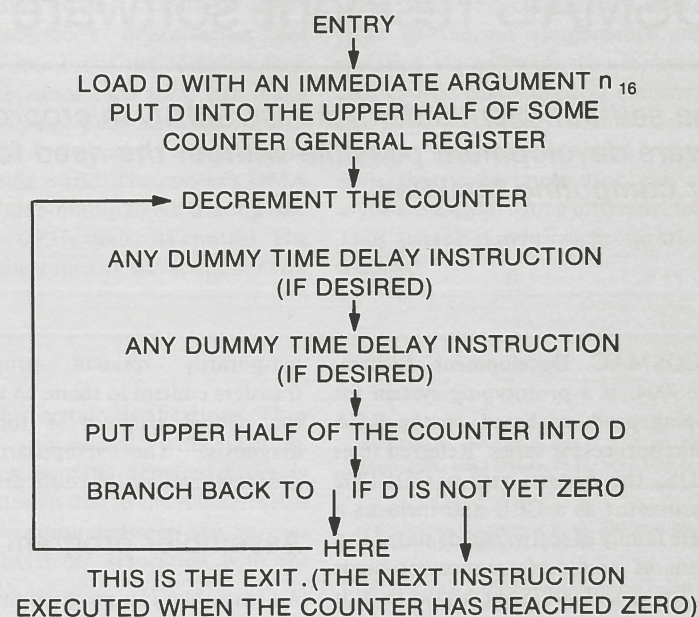


Fig. 3
Program in flowchart form, but using specific COSMAC instructions.

manner; it converts source programs written in COSMAC Level I assembly language into executable (hexadecimal) machine code.

An assembler permits the programmer to write programs using a convenient set of symbols and expressions. An input or source program consists of a sequence of statements; the assembler normally translates these statements into an equivalent sequence of hexadecimal digits (a single machine instruction or a data field of user-defined constants). This code is then placed in its proper position (i.e., assembled) in an output or object file—the executable machine program. Some statements are special control commands to the assembler. Called assembler directives, they do not directly cause output code to be produced.

CRA is a two-pass assembler. That is, it normally reads the complete source file twice to complete an assembly. During the first pass, it constructs the symbol table in RAM and prints it on the terminal, and also flags syntactic errors. On the second pass, it generates object code using the symbol-table values just derived and then prints an assembly listing. Additional program errors (for example, the detection of undefined symbols) may be flagged on the second pass.

By way of example, let us examine a simple "time-out" program that takes a number and decrements it to zero. The time from ENTRY to EXIT is approximately n times the time for one pass through the loop. Fig. 1 is a flowchart for such a program; Fig. 2 is an acceptable machine-language statement for it. The flowchart, of course, is much more understandable than the machine-language version of the program.

Understanding the resident assembler—from flowchart to operation mnemonics.

The time-out test program given above can illustrate some of the essential properties of the COSMAC resident assembler by starting from the flowchart and proceeding toward the hex form "by hand." Fig. 3 is a version of the program expressed in terms of specific COSMAC instructions.

Using shorthand mnemonics for the instructions and comments gives us Fig. 4, where:

LDI = load immediate
PHI = put high
DEC = decrement
GHI = get high
BNZ = branch if not zero

Their equivalent hexadecimal codes (for example, 3A for BNZ) can be found in the

COSMAC User Manual MPM-201. Each line of the program is now beginning to resemble an assembly-language statement.

Mnemonics and comments simplify work for the programmer.

The last version illustrates two fundamental properties of an assembly language—the use of operation mnemonics and the use of comments. An assembler is designed to recognize operation mnemonics, which are much more descriptive to the programmer, and to convert them into their hexadecimal code equivalents. In addition, an assembler is designed to ignore comment text fields in statements when it recognizes their existence. In the program version above, every comment begins with a double period (..) and extends to the end of the line. Comments are valuable to the programmer because they permit him to add documentation to a program's statements.

The assembler also assigns address locations.

The next problem considered is assigning addresses—specifically, the branch address in the last instruction in the loop. Clearly, the address assigned depends on where the program will reside in memory while it is executing. If this location is not presently known absolutely (for example, because

the routine's exact location within a larger program may change), a labeling procedure may replace the arrowed path shown. Two examples of labeling are given below in Figs. 5 and 6.

An assembler permits locations within a program to be identified by English-like symbols (e.g., "LABEL" above). Then, any reference to a location may be made by use of its label (e.g., "BNZ LABEL"). The programmer is free to select almost any sequence of characters for each label. Typically, he chooses a symbol that has some logical meaning within the context of his program (e.g., LOOP, DELAY, TEST1, SEARCH, etc.). During the process of translating the program's statements, the assembler keeps track of the addresses of all the bytes it generates. It starts from some known address reference, such as zero, using an internal location counter for this purpose. Whenever it encounters a LABEL statement, it enters the address of the instruction in a symbol table. All references to that label may then be replaced with appropriate address bytes.

An assembler also normally permits addressing relative to the position at which the reference is found. Often, the special symbol "*" refers to the address of this statement and m is a count of the number of bytes from this point.

Two forms of symbolic addressing have been defined. Fig. 7 shows how the "time-out" program now looks using statement labels; it is almost a correct assembly language program. (Three statement labels have been specified, but only one is presently referenced.)

The assembly-language equivalent of our program.

Next comes the selection of the value for n , the selection of the COUNT register, and the "DUMMY" instruction. To get the maximum delay, the original version of this program used a hex FF for the immediate byte. The assembly-language statement LDI #FF will translate properly. This selection shows that there are still many places in a program where explicit values are specified by the programmer. The "#" indicates the presence of an explicit hex constant. One can similarly explicitly identify the general register to be used as the counter with statements such as PHI 1, DEC 1, etc., assuming R1 was chosen. Suppose, however, that one wished to defer or later modify register assignments. A

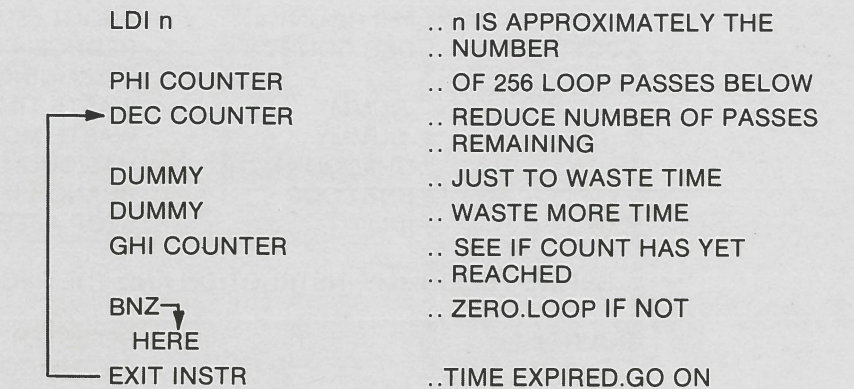


Fig. 4
Mnemonics are a bridge between the programmer's flowchart and machine language.

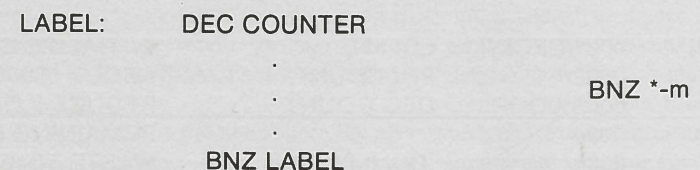


Fig. 5
Labeling replaces the arrows in Fig. 4.

Fig. 6
Addressing method counts a number of bytes past the location of this reference.

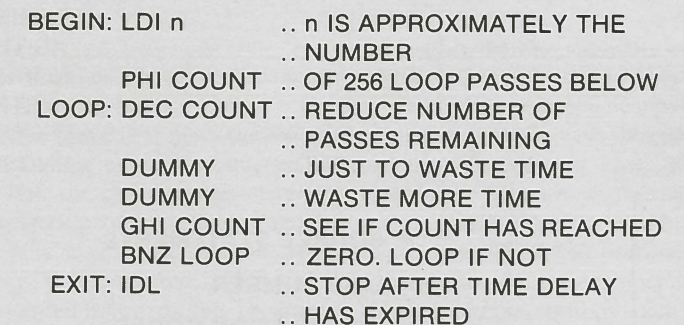


Fig. 7
Almost there: assembly-language program does not have all its statement labels referenced.

convenient permissible procedure is to continue to use the symbol as an identifier (in this case, not of a memory location, but of a general register) and give the symbol a value with a special EQUATE statement of the form COUNT=1. In this case, all occurrences of COUNT will be replaced with 1 by the assembler. If, later, one wished to reassign registers, a change to COUNT=10, for example, would automatically change all references to COUNT to the value #0A(hex). To generate a delay, one may use the NOP instruction or any other time-wasting instruction. The hex program originally given merely repeated the GHI 1 instruction three times. There are several ways by which this instruction can be expressed to


```

BEGIN:      LDI #FF      .. INITIALIZE COUNTER
            PHI COUNT    .. ABOUT 65000 PASSES.
LOOP:      DEC COUNT    .. REDUCE # OF PASSES
            .. REMAINING BY 1.
            ,DUMMY      .. WASTE TIME.
            ,DUMMY      .. WASTE MORE TIME.
            GHI COUNT   .. HAS COUNT REACHED ZERO
            BNZ LOOP    .. BRANCH IF NOT
EXIT:      IDL          .. STOP AFTER TIME DELAY.
..
.. DEFINE THE DUMMY INSTRUCTION AND THE REGISTER
..
COUNT= 1      .. REGISTER 1 WILL BE USED
               .. AS THE COUNTER
DUMMY= #91     .. NOP IS A REPEAT OF A GHI
               .. INSTRUCTION

            END          ..END OF ASSEMBLY PROGRAM

```

Fig. 8
Complete assembly-language program.

```

!M
0000 F8FF;    0001 BEGIN:  LDI #FF      .. INITIALIZE COUNTER
0002 B1;      0002      PHI COUNT    .. ABOUT 65000 PASSES.
0003 21;      0003 LOOP:  DEC COUNT    .. REDUCE # OF PASSES
0004 ;        0004      .. REMAINING BY 1.
0004 91;      0005      ,DUMMY      .. WASTE TIME.
0005 91;      0006      ,DUMMY      .. WASTE MORE TIME.
0006 91;      0007      GHI COUNT   .. HAS COUNT REACHED ZERO
0007 3A03;    0008      BNZ LOOP    .. BRANCH IF NOT.
0009 00;      0009 EXIT:  IDL          .. STOP AFTER TIME DELAY.
000A ;        0010 ..
000A ;        0011 ..DEFINE THE DUMMY INSTRUCTION AND THE REGISTER
000A ;        0012 ..
000A ;        0013 COUNT= 1      .. REGISTER 1 WILL BE USED
000A ;        0014      .. AS THE COUNTER
000A ;        0015 DUMMY= #91   .. NOP IS A REPEAT OF A GHI
000A ;        0016      .. INSTRUCTION
000A ;        0017 ..
000A ;        0018      END          ..END OF ASSEMBLY LANGUAGE PROGRAM
0000

```

SOURCE STATEMENTS
 LINE NUMBER
 INSTRUCTION CODES
 LOCATION OF INSTRUCTIONS

Fig. 9
Program listing after assembly gives machine-language instructions along with assembly-language version and the programmer's comments.

the assembler; one method uses another form of EQUATE statement to give a value to a symbol. As will be explained later, a comma may be used to precede many kinds of "constants," some whose values are explicitly stated and some whose values are derived by the assembler. The statement "DUMMY," for example, will cause a substitution of the value for the symbol. Thus, if another statement DUMMY=#91 is supplied, all occurrences of DUMMY

will be replaced by a hex 91 (which is a GHI 1 instruction). Finally, the assembler begins assigning address values starting with zero. Fig. 8 shows the final form of one assembly-language equivalent of the hex program we started with.

When this program is assembled, the listing of Fig. 9 is generated. Note the correspondence of the instruction codes in the second column to those listed in Fig. 2.

Editor program

After the user has hand-written his first COSMAC assembly-language program and wants to assemble it, he immediately faces the problem of converting the hand-written source file into a machine-readable form. This conversion involves a keyboard-to-tape operation in which lines on the coding sheet are transcribed to become lines on a source media, a floppy disk for

example. It is more likely, however, that the COSMAC resident editor will be used at this point to create the source file. Using the editor assures that the created files are in proper format for later reading by the assembler and for later modification, if necessary, by the editor.

Once a source file has been created and a first assembly-run made, it is very likely that the CRA will return error diagnostics, asking for corrections to the source file to have the program conform to CRA's rules. Typically, the changes required at this point are "trivial" but necessary. For example, spaces may have to be removed in one or more expressions, the same symbol may have been erroneously used for two purposes, an operation mnemonic may have been misspelled, or a punctuation character may have been omitted. The number of possible trivial errors is clearly large.

The editor is used to correct the errors and alter the source file to conform the program to the CRA's rules. Typically, modifications at this point merely involve inserting and deleting single characters or replacing a small string of characters by a substitute string. The erroneous source file is used as an input to the editor and the user generates a corrected source file as an output. The new file is then re-assembled. At this point, other trivial errors that were not apparent to CRA on the first run may appear. For example, an erroneous instruction operand may not have been flagged on the first assembly because its associated statement label or operation mnemonic may have also been in error. Thus, a new edit-reassemble pass may be necessary. Finally, a program is developed to which CRA does not object. At this point, a first run can take place.

The probability of a logical error in the program depends on the length of the program and the previous experience of the programmer. Assuming one or more logical errors are found (via some "debugging" procedure), the source file must again be modified. Often such modifications are no longer trivial. For example, it may be necessary to find all instructions that branch to a given location and precede some of them with one or more instructions currently not in the program. Often, it may be necessary to delete or insert some code or move some code to a different point in the program. Several duplicated sets of in-line instructions may have to be removed and replaced with calls to one common

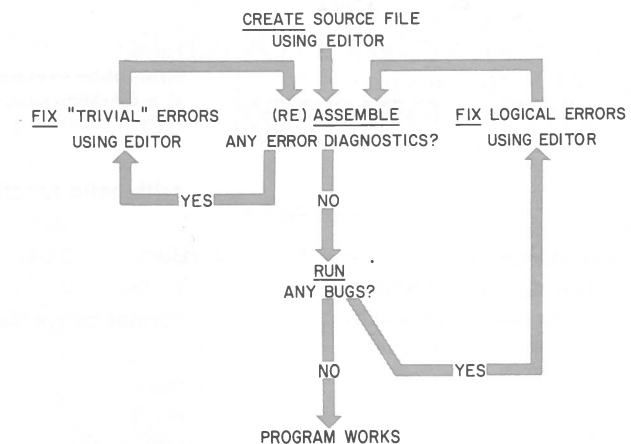


Fig. 10
Steps along the way to a bug-free program.

subroutine that must be added. The user may decide to "clean up" the program logically, in any one of several ways, or to improve its "readability" by modifying its comments or statement formats (by inserting TABs or SPACEs, for example).

Such modifications to the source file also involve the editor. After they are completed, a reassembly may again turn up new errors of the "trivial" variety, and so on. Thus, the generation of a bug-free program typically involves the process shown in Fig. 10. It is quite likely that the amount of time spent "conversing" with the editor will be much larger than that spent with the assembler.

A source program may be viewed as a long sequence of characters. When the COSMAC resident editor reads the source file, it places this character-sequence in memory, with the code in each memory byte representing one source-program character. The user is then free to type commands to the editor to manipulate the memory representation of the program. For example, the user may identify a specific location and define a character sequence to be inserted there. He may also identify certain characters to be deleted or altered. He may ask the editor to search for the occurrence of specific character sequences, after which further memory modifications (corrections) may be made.

After he is satisfied that the new memory representation of the file contains all of the changes he desires (frequently an editing session begins with a handwritten list of changes) the user asks the editor to write (create) a new output tape containing the new version of the program. This new file is then used as the input file for a reassembly.

A syntactically "correct" program that is acceptable to an assembler, may still contain logical errors that the assembler is not designed to detect. These errors can only be isolated by running the program to determine where it is malfunctioning. One common debugging procedure is to stop program execution at appropriate logical points and examine (or "dump") the contents of selected registers. It is also useful to reinitialize ("patch") the contents of selected registers before resuming execution at a desired point. Alternatively, the contents of selected registers may be printed ("traced") while the program is executing.

A programmer may explicitly include code to facilitate debugging directly in his program. For example, he may insert stop commands and/or calls to a print routine at various key points. Later, after all the errors have been found, these instructions may be removed or replaced by a "no-op" instruction. Such a process, however, assumes that the programmer can identify those program sections that are error-prone. Furthermore, the procedure is inconvenient and inefficient.

A preferred method is to use a debugging aid, which is generally defined as a facility that permits a programmer to isolate program errors without resorting to special program-preparation steps. This way the programmer is not required to predict beforehand where potential errors may lie.

Program debugging

The CSDP system, which runs on a remote timesharing computer, includes a COSMAC simulator that mimics the real hardware exactly and also includes a rich

set of debugging facilities. (See the *Timesharing Manual for the RCA CDP1802 COSMAC Microprocessor*, MPM-202.) These facilities permit a user to debug a program prior to its running on the actual hardware.

The COSMAC Development System's debugging facility (abbreviated ODB, for on-line debugging) is designed to provide many of the debugging features offered by the COSMAC Software Development Package (CSDP). CSDP is a versatile interactive aid for developing and checking COSMAC programs; it uses CSDP-augmented timesharing or in-house computer facilities. ODB, however, operates directly on the CDS in a stand-alone manner. See the *Operator Manual for the RCA COSMAC Development System*, MPM-208, for a description of ODB.

Floppy disk system

As mentioned earlier, the COSMAC Floppy Disk System (FDS), CPD18S800, is a mass-storage unit designed to work with the CDS to facilitate rapid program development. It includes interfacing hardware, special software for loading programs from the floppy diskette into memory, and special versions of the COSMAC resident editor, assembler, and utility programs.

The ROM set provided with the FDS contains a disk utility and monitor program; the system diskette contains the following programs:

- an enhanced assembler program
- an enhanced editor program
- diagnostic programs
- file movement programs
- diskette copying program
- memory save program
- demonstration program

Additional new programs for the FDS are being continuously developed. For example, a new higher-level system for the FDS will be available soon.

The outstanding feature of the floppy-disk software aids is their extreme speed advantage over those provided for use with teletypewriter terminals. For example, the assembly of a one-kilobyte program takes approximately ten minutes with the FDS, but over an hour with a teletypewriter terminal. The floppy disk's speed advantage is very important because this loading

Table I
Subroutine execution times for the CDP1802. Measurements are at a 6.4-MHz clock rate and are rescaled with a change in the system clock rate.

Arithmetic functions				
	Add	Subtract	Multiply	Divide
Best	0.042	0.040	0.872	1.405 ms
Worst	0.070	0.080	1.320	1.830 ms
Format conversion				
	Binary-BCD		BCD-Binary	
Best	1.366		0.097 ms	
Worst	2.897		0.834 ms	

and running of the assembler program takes place many times in the course of application software development.

Subroutine library

In many programs there is often a degree of commonality. To help the COSMAC user develop his program with the least amount of "reinvention" possible, a well documented program library is available. The manual *Binary Arithmetic Subroutines for RCA COSMAC Microprocessors*, MPM-206, for example, describes an extended-precision arithmetic package.

The Binary Arithmetic Subroutine Package is a set of 16-bit 2's-complement arithmetic subroutines designed to operate on COSMAC CDP1802 microprocessor systems, including the COSMAC Development System. The subroutines are coded in Level I assembly language and require 1 kilobyte of memory space; they are also available in source language on a floppy diskette for use with the RCA Floppy Disk System.

A total of 31 subroutines are included: 15 are binary arithmetic subroutines, 14 are utility subroutines, and 2 are for format conversion.

The *arithmetic functions* included in the package are 16-bit 2's-complement addition and subtraction, 16-bit 2's-complement multiplication yielding 32-bit products, and 32-bit 2's-complement division yielding 16-bit quotients and remainders.

A set of special *utility subroutines* allows the user to save and restore a group of registers on a stack or at a user-defined

RAM area. These registers are used by the arithmetic-function subroutines to store an operand and point to operands in memory. Other utility subroutines compare 16-bit operands and indicate if a register is greater than or equal to an operand.

The two *format-conversion subroutines* in the package are for interfacing the system to BCD-oriented peripheral hardware. These subroutines provide BCD-to-binary and binary-to-BCD conversions.

The Standard Call and Return Technique, described in the *User Manual*, MPM-201, can be used for all the subroutines.

Table I gives time measurements at a 6.4-MHz clock rate for the best and worst cases of the various arithmetic and format conversion subroutines for the CDP1802. The timing, however, is rescaled by a change in the system clock rate.

A 32-bit *floating-point software package* is now under development. It is intended for use in applications that require high precision in numbers of very large or small magnitude. In addition to the standard four functions (addition, subtraction, multiplication and division) there are also subroutines to perform trigonometric, inverse trigonometric, square-root, and log functions. The total package is less than 2 kilobytes long and will be available in source form. A complete manual will be issued with its release.

Summary

In summary, the COSMAC Development System is supported by a compatible family of resident software aids. Each is designed to help automate the generation of software required by products involving the RCA1800 microprocessor series.

COSMAC support literature: keeping users informed

One reason for COSMAC's acceptance is the extensive body of understandable literature supporting the microprocessor and its associated development systems.

C.A. Meyer

Because many of the potential users of microprocessors have had limited experience with computers, the microprocessor manufacturer must provide a broad range of supporting literature, from the fundamental to the advanced. Recognizing this need, the SSD Microprocessor Products activity has been supporting a well-organized program for the preparation and publication of a large amount of technical literature dealing with microprocessor hardware, software, and firmware. Moreover, it is continuously updating this literature so that users can keep up with the state of the art and thus be able to make optimum use of microprocessor-based systems.

Introductory support literature

Early in 1976, the CDP1802, a one-chip CMOS 8-bit register-oriented central processing unit, was announced. With it was introduced a family of memory and I/O support components, hardware, and software support systems suitable for testing and developing new microprocessor applications, an extensive data sheet for each new component, and a brochure (2M1137) describing the overall program. Part of the early planning for this program included the development of a comprehensive manual written to help users through the maze of new hardware, new software, and new systems.

User Manual—the best place to start.

Such a manual, the *User Manual for the RCA CDP1802 COSMAC Micro-*

Final manuscript received December 10, 1976.



processor, MPM-201, was issued within a few months of the new microprocessor announcement. This manual, written for electronics engineers having only a limited familiarity with computers and computer programming, guides the reader gently through the microprocessor architecture and introduces a set of comprehensive, easy-to-use programming instructions. The 115-page, well-illustrated (125 figures) manual provides a detailed guide to the application of the CDP1802 microprocessor in a wide range of stored-program computer systems, both special and general-purpose.

Many examples are given to illustrate the operation and use of each of the CDP1802's 91 instructions and so aid the design engineer in developing simpler and more powerful products using the wide range of microprocessor capabilities. For systems designers, this *User Manual* illustrates practical methods of adding external memory and control circuits. It also has detailed examples of the use of I/O interface lines, including DMA and interrupt inputs, external-flag inputs, command lines, processor state indicators, and external timing pulses.

The manual covers various programming techniques, with examples, as well as more advanced topics, such as the use of subroutines, interrupt service, and RAM and register allocation.

The *User Manual* is probably the most suitable entrance point for those interested in learning COSMAC microprocessor fundamentals.

Systems support literature

Three major systems, available since the introduction of the CDP1802, help simplify microprocessor programming and application. Two are a combination of software and hardware; one is software only. These three systems, the Evaluation Kit, the COSMAC Development System (CDS), and the COSMAC Software Development Package (CSDP), each have their own comprehensive technical manuals.

Use the *Evaluation Kit Manual* to learn all you need to build prototypes.

The *Evaluation Kit Manual for the RCA COSMAC Microprocessor*, MPM-203,

describes how to install and use the Evaluation Kit CDP18S020 and the hardware and firmware associated with it. The Evaluation Kit provides the key hardware and firmware elements for a computer system based on the CDP1802 Microprocessor. With this kit, augmented with an input/output terminal and a power supply, the user can readily prototype dedicated systems and evaluate software, components, and systems operation.

The *Evaluation Kit Manual* provides detailed information on the kit, its components, its overall configuration, and how it operates. The manual tells how the kit is assembled, how it can be used with various I/O terminals, how to troubleshoot hardware, how the various systems are designed, how they operate, how to check out software, and how to make use of the resident firmware. In addition, the manual contains a number of application notes describing the I/O and control features, the memory systems available and their utilization, and the use of resident firmware for Read and Type routines.

Because the development of microprocessor-based applications is a very dynamic activity, SSD Microprocessor Products is committed to a supporting program of application information and providing a continuous updating to Evaluation Kit users. As part of this update program, application note ICAN-6623 "Memory Address Function for the RCA Microprocessor Evaluation Kit CDP18S020," has been issued. Others are in process.

The *Operator Manual* is for more sophisticated development work.

The COSMAC Development System CDP18S004, a more extensive prototyping system than the Evaluation Kit, is covered in its technical supporting literature, the *Operator Manual for the RCA COSMAC Development System*, MPM-208. This 133-page manual is an operating guide on using a major prototyping aid for the design of hardware and software systems using the CDP1802 microprocessor. The COSMAC Development System (CDS) is specially designed and structured to provide a testbed in which the hardware and software prototypes of systems containing a microprocessor may be designed, built, and tested.

The *Operator Manual* includes a detailed description of each of the hardware modules in the CDS and an explanation of the functions available from the software supplied with the system. The hardware includes a rack-mountable card nest, a self-contained power supply, an easy-to-use control panel, and a basic set of eleven plug-in modules, including CPU, clock and control, address latch, memory bank select, 4-kilobyte RAM, I/O decoder, bus separator (2), byte I/O port, terminal interface, and monitor. The software supplied, which runs on the CDS in a stand-alone manner, provides a means for rapid coding and debugging of COSMAC programs. The software includes a resident assembler, editor, and utility program having a debugging facility.

The *Operator Manual* MPM-208 also describes input/output interfacing, memory module addressing, and many very useful special operating considerations.

Application notes keep you up to date with system add-ons and improvements.

Concurrent with the publication of the *Operator Manual*, a series of application notes has been published describing various circuits and techniques for augmenting the utility of the COSMAC Development System. Notes in print include the following:

- Programmable Interval Timer and Counter for Use with RCA COSMAC Development Systems—ICAN-6482
- Interrupt Priority Resolution Circuit for Use with RCA COSMAC Development Systems—ICAN-6485
- Adding Two-Level I/O Interface Capability to RCA COSMAC Development Systems—ICAN-6486
- Hexadecimal Display for Use with RCA COSMAC Development System—ICAN-6487
- Alphanumeric Display for Use with RCA COSMAC Development Systems—ICAN-6488
- Analog-to-Digital Converter for Use with RCA COSMAC Development Systems—ICAN-6490

• Digital-to-Analog Converter for Use with RCA COSMAC Development System—ICAN-6489

• PROM Programmer for RCA COSMAC Development Systems—ICAN-6491

• Hexadecimal Keyboard Interface for Use with RCA COSMAC Development Systems—ICAN-6516

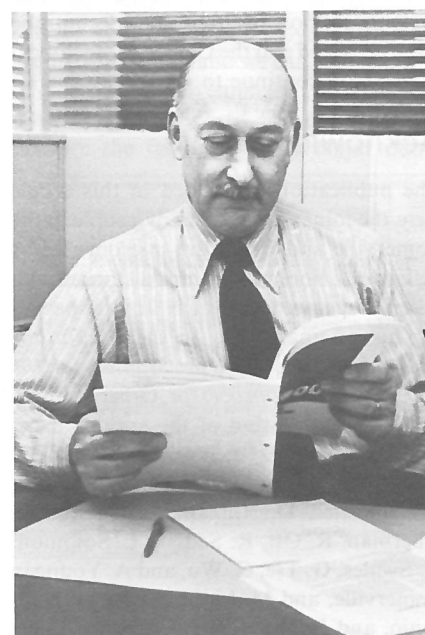
• Register-based Output Functions for RCA COSMAC Microprocessors—ICAN-6562

• Design of Clock Generators for use with RCA COSMAC Microprocessor CDP1802—ICAN-6565

• Power on by Reset/Run Circuits for the RCA CDP1802 COSMAC Microprocessor—ICAN-6581

Charles Meyer has worked on technical literature for a wide variety of technical products since joining RCA in 1946. He has worked on microprocessor manuals, application notes, and other technical publications for the Solid State Division since January 1976. He has served on the advisory board for *TREND*, as a consulting editor of the *RCA Engineer*, and an associate editor of the *RCA Review*.

Contact him at:
Engineering Publications
Solid State Division
Somerville, N.J.
Ext. 6548



For rapid program development, see the *Floppy Disk Manual*.

Another major piece of technical literature supporting the COSMAC Development System is the *RCA COSMAC Floppy Disk System* CDP18S800 *Instruction Manual*, MPM-211. This manual describes a mass-memory storage unit designed for use with the CDS CDP18S004 to facilitate rapid program development using the advantages of the floppy disk over paper-tape or cassette devices. The manual describes the hardware interfacing to the CDS, the software for loading from a diskette into RAM, and the special versions of the COSMAC resident editor and assembler programs supplied on a diskette.

The manual also describes the special operations required when the floppy disk system is used, how to install the interface card, how to install the ROM set, how to set up the drive mechanism, and what preventive maintenance care to exercise. The Disk Utility and Monitor program is also described, as well as the Loader routine and the Read and Write routines. The manual also gives information on the Disk Assembler and the Disk Editor programs, along with examples of their use. The eight appendices to this manual include program listings and other useful information.

The *Timesharing Manual* gives you the information to develop microprocessor programs without new hardware.

The third major support system for microprocessor users, the COSMAC Software Development Package (CSDP), has an instruction manual as well as other supporting literature. The *Timesharing Manual for the RCA CDP1802 COSMAC Microprocessor*, MPM-202, provides a comprehensive guide to CSDP users. CSDP, with the information provided in its supporting manual, permits the user to develop microprocessor programs without any new hardware, using only a timesharing terminal for existing in-house or leased computer facilities. Through the use of CSDP-augmented timesharing or in-house computer facilities, software and hardware can be developed concurrently.*

CSDP facilities include a cross-assembler, an error-checking simulator, and powerful debugging tools. CSDP is presently

available on commercial timesharing systems and, as a source program in FORTRAN IV, it is available with a detailed installation manual (MPM-204) for installation in most computer facilities.

The *Timesharing Manual* reviews the architecture, organization, and operation of the CPU and provides the programmer with a means of writing and modifying programs using convenient mnemonic symbols rather than machine language. The cross-assembler translates these symbols into machine language.

The COSMAC assembly language requires no fixed-field alignment, permits the use of blanks for added readability, and permits multiple instructions per line. By also permitting the use of comments on any line, it enables the program to be self-documented.

Additionally, the *Timesharing Manual* provides detailed coverage of the CSDP constituents, pertinent programming techniques, and a very valuable description of common programming "bugs." The manual describes the development of a sample program including the very important debugging steps. Six useful appendices are included in this 88-page manual, as well as a comprehensive index.

Other CSDP-supporting literature includes the *Installation Manual for COSMAC Software Development Package*, MPM-204, mentioned above, which describes how to install the CSDP in a timesharing in-house computer facility. This manual is intended for programmers acquainted with FORTRAN. In addition to step-by-step installation information, the manual describes some specific features, not standard FORTRAN, which provide additional utility and convenience to the programmer.

Other key manuals

As part of the overall planning, hardware and software are being continuously developed to augment COSMAC capabilities. In the software area, one important set of programming aids available on a floppy diskette (CDP18S826) is also covered in a manual. This

*CSDP is available to RCA users on CMS in Cherry Hill and TSO in Somerville.

manual, *Binary Arithmetic Subroutines for RCA COSMAC Microprocessors*, MPM-206, describes a set of 16-bit 2's-complement arithmetic subroutines designed to be operated on CDP1802 microprocessor systems including the COSMAC Development System. In the *Binary Arithmetic Subroutine Manual*, 31 subroutines are described. Fifteen are binary arithmetic subroutines, 14 are utility subroutines are described. Fifteen are sion. The arithmetic functions included are 16-bit 2's-complement addition and subtraction, 16-bit 2's-complement multiplication yielding 32-bit products, and 32-bit 2's-complement division yielding 16-bit quotients and remainders. The 62-page manual includes the complete program listings as well as a sample program illustrating its use.

Another software development, which will be available on a diskette and for which a manual is in preparation, is loosely referred to as the "floating-point package." This package is a group of advanced mathematical subroutines that further enhance COSMAC microprocessor capabilities.

In the hardware area, a very interesting new development is the COSMAC Microterminal CDP18S021. This data terminal is accompanied by the *Instruction Manual for the RCA COSMAC Microterminal CDP18S021*, MPM-212. This 28-page manual describes the installation and application of a portable handheld data terminal designed for microcomputer systems using the CDP1802 microprocessor. It is a low-cost, low-power alternative to a conventional data terminal such as the teletypewriter. The Microterminal, together with its associated programmed ROM, provides a means of controlling a COSMAC-based system and supplies hexadecimal I/O capability. The Microterminal is designed to interface directly with COSMAC support hardware systems such as the CDP18S020 Evaluation Kit, but it can also be readily designed into user-built systems to provide the same control, communications, and debugging functions.

The manual includes a description of the hardware and the software programs available in the ROM supplied with the Microterminal and also includes installation instructions and utility-program listings. The operating modes of the Microterminal are described as well as several examples of operating sequences.

RCA Microprocessor Manuals are available to RCA employees for their use at a considerable reduction from list price.

		List price	Employee price
MPM-201A	User Manual for the RCA CDP1802 COSMAC Microprocessor	\$ 5.00	\$3.00
MPM-202	Timesharing Manual for the RCA CDP1802 COSMAC Microprocessor	10.00	6.00
MPM-203	Evaluation Kit Manual for the RCA COSMAC Microprocessor	15.00	9.00
MPM-206	Binary Arithmetic Subroutines for RCA COSMAC Microprocessors	5.00	3.00
MPM-208	Operator Manual for the RCA COSMAC Development System	10.00	6.00
MPM-211	RCA COSMAC Floppy Disk System CDP18S800 Instruction Manual	10.00	6.00
MPM-212	Instruction Manual for the RCA COSMAC Microterminal	2.00	1.20
MPM-109	COSMAC Microtutor Manual	2.00	1.20

To purchase these manuals, send order with personal check, IPR, or department charge number to:

Publication Services,
Mail Stop 37
RCA Solid State Division
Somerville, N.J.

COSMAC fun and games

There is one more manual that merits special attention, the *COSMAC Microtutor Manual*, MPM-109. This Manual is an instruction book supplied with a very unusual piece of equipment, the CDP18S011 Microtutor. The Microtutor is a small, simple, inexpensive, and easy-to-understand computer. It comprises 256 words of memory, input switches, a two-digit output display, and the COSMAC microprocessor. The Manual tells how to use the Microtutor to learn about microcomputers with fun and games on a simple computer; it also contains a number of programs for just fooling around. And after the user is hooked on how easy it is to work with binary numbers and hexadecimal code, he is shifted smoothly into COSMAC structure and program development. Before he realizes it, he is a microcomputer expert. Not really, but it is fun to work with and quite a bit of skill and knowledge rubs off. The Microtutor manual is a 48-page gem—I wish I had written it.

Importance of technical support

In the microprocessor field, particularly, customers measure a supplier not only by

the hardware, software, and field engineering support that he provides, but also by the quality, availability, and comprehensiveness of his supporting literature. The nine manuals described here, together with the application notes and technical data sheets, already constitute a substantial library providing technical support to COSMAC microprocessor users. With the continuing development of more COSMAC hardware, software, and systems, supporting literature will continue to be provided and the COSMAC technical library will continue to grow.

Acknowledgments

The publications described in this article were the joint efforts of many people in the Somerville and Princeton microprocessor activities. Foremost among those who deserve special mention is J. Weisbecker for his original work on the *Microtutor Manual* and on the *User Manual for the COSMAC Microprocessor*. Others who, to the best information of the author, merit acknowledgments include D. Block, D. Carley, W. Clark, H. Edinger, E. Fulcher, A. Gonzales, D. Hillman, K. Karstad, J. Oberman, R. Ott, R. Sedlak, L. Solomon, N. Swales, G. Tse, C. Wu, and A. Young in Somerville, and M. Lippman, A. Marcan-tonio, and P. Russo in Princeton.

Simplifying microcomputer architecture

J. Weisbecker

In 20 years computer hardware has become increasingly complex, languages more devious, and operating systems less efficient. Now, microcomputers afford the opportunity to return to simpler systems. Inexpensive, LSI microcomputers could open up vast new markets. Unfortunately, development of these markets may be delayed by undue emphasis on performance levels which prohibit minimum cost. Already promised are more complex next generation microcomputers before the initial ones have been widely applied. This paper discusses these points and describes a simplified microcomputer architecture that offers maximum flexibility at minimum cost.

LARGE SCALE INTEGRATION (LSI) of semiconductor devices has finally become a practical reality. The long-awaited revolution in electronic products appears to be at hand. The basis of this revolution is the ability to provide complex electronics at greatly reduced prices. Major cost reduction opens up entirely new markets and is as significant a development as the invention of the vacuum tube or transistor.

The four-function electronic calculator represents the first wave of the revolution. Further new markets will emerge with the ability to provide complete stored program computers at a fraction of current minicomputer costs. A number of microcomputer chip sets have already been announced.^{1,2} We can expect a proliferation of microcomputer types and products based on them over the next several years. Unfortunately, old habits are hard to break and we can also expect to see increased emphasis on performance instead of cost.³ This could easily obscure the fact that many major new

markets will depend primarily on absolute cost.^{7,8}

Consumer, educational, small business, and communications markets are prime targets for truly low-cost microcomputer based products. The architecture described in this paper was developed to satisfy the requirements of these potential new markets. Practical, stand-alone systems (including input/output device control and memory) requiring as few as six LSI chips are feasible with this architecture. Such systems have been breadboarded and programmed. Based on this experience, the microcomputer described appears to satisfy the requirements of a much wider range of applications than originally intended. It also appears to be simpler than most existing microcomputers. It is estimated that this new architecture compares favorably with the complexity of current

four-function calculator chips. This simplicity is expected to provide significant production advantages. Improved yields and decreased testing costs are anticipated.

Since LSI improvements are permitting ever larger numbers of devices per chip, there are definite long-term advantages in minimizing microcomputer complexity. If the microcomputer is prevented from growing in complexity as the device per chip ratio improves, more of the system can be pulled back into a single chip. For example, small systems in which both memory and microcomputer are provided on a single chip become feasible resulting in added, long-term cost-reduction potential. This approach is ruled out when increased device per chip ratios are used to provide more complex microprocessors.

Joseph A. Weisbecker, LSI Systems Design, Solid State Technology Center, RCA Laboratories, Princeton, New Jersey, graduated from Drexel University in 1956 with the BSEE. Prior to joining RCA Laboratories in 1970, he was active in various product planning positions within RCA's Computer Systems Division. He has made major contributions to RCA computer development since 1953. During a three-year sojourn with a smaller company, he developed a number of customized data terminals. Mr. Weisbecker holds 19 patents with others pending. He is a member of IEEE and Eta Kappa Nu, and he has received an RCA Laboratories Achievement Award. His children have had a computer at home for several years which reinforced his interest in low-cost, widely available microcomputer systems.



Final manuscript received November 26, 1973.

Design philosophy

Minimum system cost is the primary goal. To achieve this goal, an architecture is required that is both simple and flexible. Simplified computer architecture has received relatively little attention in the literature. Prior approaches toward simplified computers appear to be incompatible with microcomputer application goals.^{4,5,9}

The architecture that was finally developed evolved from examining proposed applications. Another approach would have started with a more or less conventional minicomputer architecture and instruction set. This latter approach was discarded due to fundamental differences in minicomputer and microcomputer applications. It was also felt that a minicomputer starting point would not yield the simplest architecture.

Since a single-chip microcomputer promises minimum cost, the architecture was constrained to a 40-pin interface. Smaller microcomputer interfaces tend to require extensive multiplexing of interface signals which adds demultiplexing logic external to the microcomputer chip. This increases system cost.

An 8-bit parallel (or byte) architecture was chosen. This yields maximum performance consistent with interface pin constraints and is compatible with input/output requirements. One and four-bit organizations unduly restrict the range of potential applications. Sixteen or more bits exceed single-chip pin constraints or impose the need for multiplexed word transfers.

Since continued memory cost reduction is anticipated, techniques using memory to reduce hardwired logic complexity are heavily relied on. It is also apparent that many microcomputer applications will fall into the intelligent buffer category. For these reasons, direct memory addressing capability of up to 64K bytes is provided. Random-access memory (RAM) and read-only memory (ROM) can be mixed in any combination via a common memory interface. This is a distinct simplification over an architecture that provides separate RAM and ROM interfaces. The common RAM/ROM interface also enhances flexibility. System simplicity results since a single LSI chip containing both ROM and RAM segments will suffice for many

applications.

While low memory costs can be expected, very low-cost systems will result only from minimizing memory capacity requirements. A unique architecture was devised which uses an 8-bit instruction format. This permits compact programs and subroutines. Useful systems requiring 1024 bytes or less of memory are possible with this format.

Random control logic uses chip area less efficiently than register/memory arrays. For this reason a simple, fixed cycle, microinstruction set was developed to reduce required control logic. The user has the option of programming directly in microcode, using a set of subroutines stored in memory (ROM/RAM), or a combination of these approaches. Sets of subroutines stored in memory are equivalent to applications-oriented macroinstructions and can readily be provided where ease of programming is important. On the other hand, many systems will utilize the microcomputer as a substitute for special purpose logic and can be programmed directly and efficiently in microcode. This approach retains most of the advantages of a microprogrammed computer but eliminates much of the specialized, hardwired sequencing and control logic usually associated with microprogrammed systems.⁶ Simple, short-subroutine-calling byte sequences provide flexible macroinstruction definition.

Whether used as a component of larger systems or as a freestanding computer, the microcomputer architecture requires efficient, flexible, input/output capability. This is provided via programmed byte transfers and a built-in direct memory access (DMA) channel. Programmed input/output byte transfer instructions provide maximum flexibility for in-

put/output selection, control, and data transfer. The DMA channel facilitates high-speed I/O block transfer, TV display refresh, and initial program loading with a minimum of external logic. While the inclusion of a DMA channel adds negligible complexity to the microcomputer architecture, it greatly simplifies system design, leading to reduced overall cost. In addition to the two basic I/O modes, four uncommitted flag lines are provided for activation by external devices. These flags can be tested as required by program. A flexible program interrupt capability also exists. Individual reset and load lines minimize external initializing logic.

The overall design philosophy consisted of developing a simple, flexible, microcomputer architecture which satisfies a wide range of potential applications at minimum cost. Performance levels were chosen to satisfy large-volume applications without overkill. The resulting architecture can be implemented initially on one or two chips using slow MOS technology.

Instruction execution times in the range of 4 to 8 μ s are anticipated with LSI technologies that approach current TTL speeds. Experimental work has demonstrated that this performance level is adequate for most anticipated applications.

Microcomputer architecture

Fig. 1 illustrates the microcomputer architecture. "R" represents an array of sixteen, 16-bit general purpose registers. (This is essentially a 16x16-bit RAM.)

Registers P, X, and N are three 4-bit registers. The contents of P, X, or N select one of the 16 R registers. Register R(N) will be used to denote the specific R

register selected by the 4-bit hex digit contained in the N register. R0(N) denotes the low-order 8 bits (byte) of the R register selected by N. R1(N) denotes the high-order byte. The contents of a selected R register (2 bytes) can be transferred to the A register. The 16 bits in A are used to address an external memory byte via an 8-bit multiplexed memory address bus. The 16-bit word in A can be incremented or decremented by "1" and written back into a selected R register.

M(R(N)) refers to a one-byte memory location addressed by the contents of R(N). This indirect addressing system is basic to the simplicity and flexibility of the architecture.

Register D is an 8-bit register that functions as an accumulator. The arithmetic logic unit (ALU) is an 8-bit logic network for performing binary add, subtract, logical "and," "or," and "exclusive or" on two 8-bit operands. One operand is the bus byte and the other is contained in the D register. The D register can also be shifted right by one bit position. Add, subtract, and shift operations set a 1-bit overflow register (not shown) which can be tested by branch instructions.

The I is a 4-bit instruction register. Four-bit operation codes are placed in I and decoded to control instruction execution. Bytes can be read onto the common data bus from any of the registers, external memory, or input/output devices. A data bus byte can, in turn, be transferred to a register, memory, or input/output device.

The operation of the microcomputer is best described in terms of its instruction set. A one-byte instruction format is used as shown in Fig. 2.

Each instruction requires two machine cycles. The first cycle causes an 8-bit instruction to be fetched from external memory and placed in the I and N registers. This is written as M(R(P))→I,N. The 4-bit digit in register P selects R. R(P) is then transferred to A and used to address memory. While waiting for the

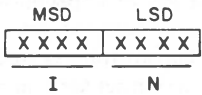


Fig. 2 — Eight-bit instruction format.

Table 1 — Instruction set for the microcomputer.

Register Operations	
Hex digit	Operation
[1]	Increment R(N) by 1
[2]	Decrement R(N) by 1
[8]	Transfer R0(N) to D
[9]	Transfer R1(N) to D
[A]	Transfer D to R0(N)
[B]	Transfer D to R1(N)
[C]	Transfer D0 to R00(N)
Memory Operations	
[4]	Load D from M(R(N)) and Increment R(N)
[5]	Store D in M(R(N))
Miscellaneous Operations	
[0]	Idle
[3]	Branch
[6]	Input/output byte transfer
[7]	Interrupt control
[D]	Set P to value in N
[E]	Set X to value in N
[F]	ALU operations

memory access, A is incremented by "1" and replaces the original contents of R(P). The most significant digit of M(R(P)) is placed in register I and the least significant digit is placed in register N. At the end of an instruction fetch cycle, I and N always contain the 8-bit instruction originally addressed by the current program counter [R(P)], and this program counter has been incremented to point to the next memory byte in sequence. Note that any R register can be selected as the current program counter by merely changing the digit in register P. Multiple program counter systems and simple branch and link operations are readily achieved with this approach.

The next machine cycle always causes the instruction contained in registers I and N to be executed. This fixed two-cycle, fetch-execute sequence simplifies control logic and permits program interruption or DMA cycle stealing to occur only between instructions. This results in even further control simplification. Because the operation code in register I is limited to 4 bits, only 16 instruction types need be decoded. The 16 possible operations specified by the hex digit in I are listed in Table I.

The first group of instructions permits selecting any 16-bit general purpose

register (R) and incrementing or decrementing it. Upper or lower halves of selected R registers can be copied into D or set from D by these instructions. Operation "C" permits the least significant 4 bits of D to be set into the least significant 4 bit positions of any R register. This facilitates table-lookup operations using 4-bit digit arguments.

The two basic memory operations permit loading D from memory and storing D in memory. Used in combination with the register operations, selected general purpose registers can be set or stored. Instruction "4" is of particular interest. When N equals P, this instruction permits a byte to be retrieved directly from the program stream and placed in D. Since R(N) is the program counter, incrementing it maintains program counter integrity. A three-byte sequence serves to set a one-byte constant into any R register. This technique is normally used for initialization of R registers.

The last group of operations provides a variety of functions. The idle state can be entered via program or a reset line provided in the microprocessor interface. The idle state waits for externally generated program interrupts or DMA requests. The branch instruction performs a test specified by the value in N. As a result of this test, the next byte in memory, as addressed by R(P), is either skipped or placed in the lower half of R(P). This latter action causes a branch within the current 256-byte memory segment. Tests specified by N include zero in D, the states of four externally activated flags, and the status of the ALU overflow register. Two instructions, "D" and "E", permit the current digit in the P or X register to be modified. The "D" instruction provides the ability to change program counters at any point in a program. For example, "D4" would immediately change the current program counter to R(4). Branch and link operations are thereby facilitated. The "E" instruction permits changing the ALU operand or input/output byte address pointer. Instruction "F" permits 8-bit ALU operations. N designates the specific ALU operation to be performed. One of the operands comprises the byte contained in D. The other operand comes from memory. The second operand can be addressed by either R(P) or R(X) as specified by N. The result of ALU operations always replaces the original byte in D.

Instruction "6" permits byte transfers between memory and input/output devices via the common byte bus. The value of N specifies the direction of the byte transfer. $M(R(X))$ can be sent to an input/output device or any input/output device byte stored at $M(R(X))$. In the former case $R(X)$ is incremented permitting X to be set equal to the current P value. The digit in N is made available externally during execution of the input/output byte transfer instruction. This digit code can be used by external I/O device logic to interpret the common bus byte. For example, specific N codes might specify that an output byte be interpreted as an I/O device selection code, a control code, or a data byte. Other N codes might cause status or data bytes to be supplied by an I/O device. In small systems the N code can directly select and control I/O devices.

Four flag lines that can be activated by I/O devices are provided. These can be used as general purpose I/O device status indicators (byte ready, error, etc.) These flag lines are tested by the branch instruction. Two interface lines control the built-in DMA channel. An I/O device can activate either an input or an output byte request line. At the end of the execution of the current instruction, a DMA channel cycle will occur causing the requested memory I/O device byte transfer to occur. $R(0)$ is used for addressing memory during DMA cycles and is automatically incremented by one, following each byte transfer. Once initiated, blocks of data can be efficiently transferred between an I/O device and memory, independent from normal program execution.

A program interrupt line can be activated at any time by external means. At the end of the current instruction, an interrupt cycle will occur. During this cycle, X and P are placed in an 8-bit temporary storage register (T); P is then set to 1 and X is set to 2. Normal fetching and execution is then resumed. Activation of the interrupt line therefore causes a branch to the instruction stream addressed by $R(1)$. $R(2)$ should point to a memory area used by the interrupt routine to store the state of the machine for subsequent return from interrupt. Instruction "7" with N equal to 8 stores the contents of T in the memory location specified by $R(X)$. It is a "save state" type instruction. If N is 0, instruction "7" causes $M(R(X))$ to be placed in P and X. $R(X)$ is incremented and an interrupt mask bit is reset. This

instruction provides a "return after interrupt" function. The interrupt mask bit inhibits further responses to external activation of the interrupt line. This mask is always set by an interrupt permitting multiple interrupts to be queued under program control.

Programming considerations

Since the instruction set of this microcomputer differs considerably from that normally encountered, some comments relative to programming are in order.

A major difference between this architecture and more conventional organizations lies in the complete separation of operation codes and memory addresses. Conventional instructions have one or more addresses associated with each operation code. This system utilizes a limited table of memory addresses contained in a set of general purpose registers. These registers may also be used for program counters and data storage. Their use is entirely controlled by program, with the exception of $R(0)$, $R(1)$, and $R(2)$. This structure is basic to the simplicity and flexibility of the architecture. It also permits the use of a short, 8-bit instruction format resulting in compact coding.

It has long been realized that storing a full memory address with each operation code is inefficient since a small number of unique memory addresses are repeated many times throughout a program. Various abbreviated addressing schemes have been used to permit more compact programs. These are almost always offered as optional alternatives to providing a full address in each instruction. Here we must always obtain a memory address from the limited, current set in the 16 general-purpose registers. We might intuitively suspect that this would be an unduly restrictive approach. Surprisingly, it turns out to be relatively easy to write programs and is highly efficient relative to the amount of memory used. A variety of programmers, from those who have only used high-level languages to engineers with limited programming experience, have had little difficulty in adapting to this architecture.

A number of programs have been written for a breadboard version of the microcomputer with a variety of in-

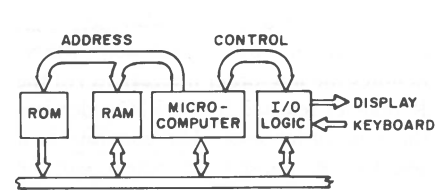


Fig. 3 — Calculator system.

put/output device attachments. This experience has validated the flexibility and efficiency of the architecture. For example, a four-function calculator program was found to require only 1024 bytes of memory, most of which could be ROM. This included provision for keyboard input; operands up to 30 digits; TV display refresh storage; 5x7 digit font conversion table; push-down stack; and algorithms for signed, n -digit decimal add, subtract, multiply, and divide. An interpreter for a simple, decimal, tutorial language was written in less than 600 bytes. A number of game and/or educational programs require well under 1000 bytes of memory. Many small business and communications systems programs are possible with 2000 to 4000 bytes of memory.

While the instruction set initially appears quite limited, it should be kept in mind that each operation requires only one byte of storage (ROM or RAM). Short sequences of these microinstructions readily provide macro-operations.

Apparent weaknesses in the architecture are the limited branch capability (within 256 bytes) and the lack of a hardwired program stack for multilevel nested subroutines. These apparent oversights are the result of a deliberate design philosophy which eliminates special purpose logic for those functions which are performed easily by subroutines. The architecture permits a flexible subroutine "call" and "return" system requiring less than 70 bytes of memory. This includes a push down stack for nested subroutines. By providing this system in software (or firmware) it can be tailored to individual applications.

Where extensive programming effort is required, a set of applications-oriented subroutines is easily developed. These subroutines constitute a user-oriented macroinstruction set for minimum effort programming. This technique has proved extremely useful in an experimental small

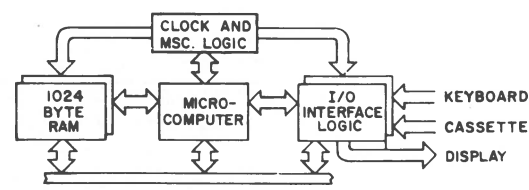


Fig. 4 — Six-chip, stand-alone system.

business system.

For microcode programming, an assembly language has been developed. This approach simplifies machine language coding considerably. An interactive simulator greatly facilitates initial program debugging. Both of these microcomputer software support systems are readily modified to run on existing time-sharing systems.

In general, the simplified microcomputer presents no difficulty in programming. It provides a simple set of short, easy to understand microinstructions that do not require high skill levels to use. For specific applications, tailored macroinstructions are readily provided via a flexible subroutine handling system.

Typical systems

Several systems using the microcomputer can be outlined. Many others are, of course, possible.

Fig. 3 indicates a possible microcomputer-based calculator. ROM and RAM might be provided on one chip resulting in a basic three-chip calculator. Functions could easily be added with ROM increments. This type of system could also provide a programmable calculator.

Fig. 4 illustrates a stand-alone system which might require only six LSI chips total.

It is assumed that 4x1024-bit memory chips will be available within the next several years. Subsequent LSI improvements could further reduce the chip count. Use of a small keyboard, audio cassette,^{10,11} and CRT display might reduce system cost to a few hundred dollars. Such a system could have wide application in consumer and educational markets. This system, with more memory, hardcopy output, and low-cost

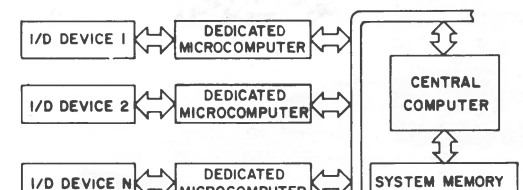


Fig. 5 — Dedicated multi-microcomputer system.

floppy disk (or magnetic bubble bulk storage), would provide the basis for a wide range of inexpensive, turnkey, small business systems.

Fig. 5 illustrates a large-computer system in which each I/O device is controlled by a dedicated microcomputer providing an intelligent buffer, as well as a replacement for special purpose logic. RAM, ROM, microcomputer, I/O device and central computer interface circuits could readily be provided on a small set of LSI chips. The microcomputer DMA feature is extremely useful for high-speed block transfers in this type of system. Downline loading of the microcomputer memory can immediately change its mode of operation. Off-line editing and maintenance is provided free. This type of large-scale system approach will become more popular in the future as microcomputer costs decrease.

The performance level of the simplified microcomputer described is more than adequate for the above types of systems as well as many others.

Conclusions

Much current microcomputer development effort appears to be directed toward improved performance. There is, however, a need for simple, minimum cost structures that will satisfy large-volume applications which do not require minicomputer performance levels. These microcomputers must also be organized to reduce total system cost. One such microcomputer architecture has been developed. It promises low cost, together with minimum external memory and system logic requirements. Hopefully, microcomputers of this class will accelerate the development of major new markets.

Currently high input/output device costs might be used as an argument against minimizing microcomputer cost. This is

extremely short-sighted. The availability of ten-dollar microcomputer chips will, by itself, exert considerable pressure on the development of compatible low-cost I/O and bulk storage devices. Even now there are many potential new products that demand minimum cost microcomputers of the type described.

Because of its flexibility and potential for low-cost systems, RCA is currently developing a COS/MOS-LSI version of this microcomputer — COSMAC, SOS versions are also being investigated for applications requiring higher instruction execution rates. Both implementations are expected to find wide application in a variety of future products.

Acknowledgments

The following people have devoted considerable effort toward evaluating and developing software for the microcomputer described here: S. Heiss, A.R. Marcantonio, J.T. O'Neil, A.D. Robbi, P.M. Russo, R.O. Winder, and C.T. Wu. Without this effort, validation of the architecture would have been impossible.

References

1. Lapidus, G., "MOS/LSI Launches the Low-Cost Processor," *IEEE Spectrum* (Nov. 1972) pp. 33-40.
2. Sideris, G., "Microcomputers Muscle In," *Electronics* (March 1, 1973) pp. 63-64.
3. Hoff, M.E., Jr., "Applications for Microcomputers in Instrumentation," 1973 IEEE Intercon Papers, Session 21/1.
4. Hitt, D.C., Ottaway, G.H., and Shirk, R.W., "The Minicomputer—A New Approach to Computer Design," *Proc. of 1968 Fall Joint Computer Conference*, pp. 655-662.
5. Conn, R.W., "The Dinkiac I," *Proc. of 1971 Spring Joint Computer Conference*, pp. 1-9.
6. Reyling, G., Jr., "LSI Building Blocks for Parallel Digital Processors," 1973 IEEE Intercon Papers, Session 21/3.
7. "Backer Sought for Information Center," *Electronics* (Mar. 29, 1973) pp. 33-34.
8. "Personal Lifetime Computer Foreseen," *Electronics* (Sept. 11, 1972) pp. 40-42.
9. KENBAK-1 Computer Programming Reference Manual, KENBAK Corp. (April 1971).
10. "Putting Data on an Ordinary Audio Recorder," *The Electronic Engineer* (May 1972) p. DC-9.
11. Wolf, E., "Ratio Recording for Lower Cassette Recorder Cost," *Computer Design*, (Dec. 1972) p. 76.



Photo 1: The COSMAC VIP system demonstrating its graphics abilities. The kit version of this product includes the board alone; the user must supply a standard video monitor (or modified TV) and inexpensive tape recorder.

COSMAC VIP, the RCA Fun Machine

This article first appeared in the August 1977 issue of BYTE.
© 1977 BYTE Publications Inc, Peterborough NH USA.
All rights reserved. Reprinted by permission.

Joseph A Weisbecker
1220 Wayne Av
Erlton, Cherry Hill NJ 08002

Most beginners think of computers only in terms of solving equations or processing data in English language format. Ask someone with limited exposure to computers about their use in homes and they invariably mention kitchen recipe files, meal planning, income tax preparation or checkbook balancing. I don't think any of these applica-

tions has really caught on so far. This view of computers as super calculators or data processors is apparently shared by many people currently involved in the home computer movement. This orientation keeps prices higher than necessary for the majority of potential hobby users who could have just as much fun with much less sophisticated machines.

The COSMAC VIP was created to provide a viable alternative to expensive hobby computers. It was not designed to compete

with existing systems, but rather to maximize the number of people who can afford to own their own high technology graphics and games fun machine.

In this article I'll give you some background information, outline my design philosophy, and describe the COSMAC VIP system. You can then decide whether this article represents a blatant sales pitch or an honest explanation of a relatively low cost approach to small personal computers. Home or business users who want extensive data processing capability should stop reading now. You will be better off with one of the more expensive BASIC language oriented systems. Those who are beginners or are interested in computers for fun at absolute minimum cost should continue reading.

First some background. About five years ago I developed a system called FRED. This was the prototype for a low cost micro-computer aimed at video games, hobby and school use. (FRED was described in the August 1974 issue of the *IEEE Computer* magazine.) COSMAC VIP evolved from this FRED system. My family and friends have been using this type of system for almost five years and we've only scratched the surface of applications for it. In fact, my \$2000 personal BASIC system seems dull by comparison and is gathering dust. The development of the 1 card COSMAC VIP system is due primarily to a desire to share fun I've been having with as many others as possible. RCA deserves a lot of credit for supporting the development and making the system available for home and school use.

Design Philosophy

COSMAC VIP is based on the premise that a home computer should be low cost, easy to use and fun for the whole family. It is therefore aimed at video graphics and games applications. Even the youngest family member can have hours of fun drawing pictures on the TV screen.

Let's examine some cost factors. While memory and logic costs continue to decrease, a 2 K byte programmable memory will always be cheaper than a 16 K byte one. To many designers, lower memory costs mean that they can design more memory into a system at the same price. To me it means that I can design lower cost computers using about the same amount I've always used. When the current crop of \$1000 computers drops to \$500, a COSMAC VIP type computer might drop to \$150. The lower the cost, the more people can own their own computers.

The drastic price reduction for a central processing unit caused by the development of single chip microprocessors has not been matched by equivalent cost reductions for conventional computer input and output devices. The lowest cost home computers can only be achieved if the need for specialized IO devices is minimized. I don't think it is fair to ask the limited budget, unsophisticated beginner to spend several hundred dollars on a system which uses a \$10 central processor but requires an additional expenditure of \$1000 for a terminal and enough memory to make it useful. The COSMAC VIP minimizes IO device cost by relying only on a video monitor, audio cassette recorder and 16 position key pad. CMOS circuits reduce power supply costs, and a minimum number of integrated circuits permit the entire computer, including keyboard, to be packaged on one inexpensive printed circuit board.

As I see it, four major obstacles must be overcome before everyone can own their own computers. The first obstacle involves justifying a relatively high priced piece of equipment which is often useful to only one family member. COSMAC VIP was designed for relatively low cost and includes 20 video game listings that can be immediately loaded and played by the whole family. This goes a long way toward eliminating the first obstacle.

The second obstacle is the complexity of conventional computer operating and programming procedures. I can't understand how a beginner can be expected to cope with complex operating systems. COSMAC VIP programs are initiated with a single RUN switch. Any memory byte can be examined or modified using the built-in hexadecimal keyboard. Loading programs is easy. Just enter the starting address followed by the byte sequence to be stored. No extra key depressions are required between bytes. Memory addresses and stored bytes are displayed on the TV screen in hexadecimal format. Tapping a single key steps through memory letting the user examine stored bytes without modifying them.

To load a cassette program into memory, you merely enter the starting memory address and single digit block length code via the hexadecimal keyboard and play the tape. Less than 30 seconds later, COSMAC VIP shows you the last byte loaded into memory on the TV screen and you're ready to run your program. Recording the contents of memory on a cassette is just as easy.

A third obstacle in the path of home

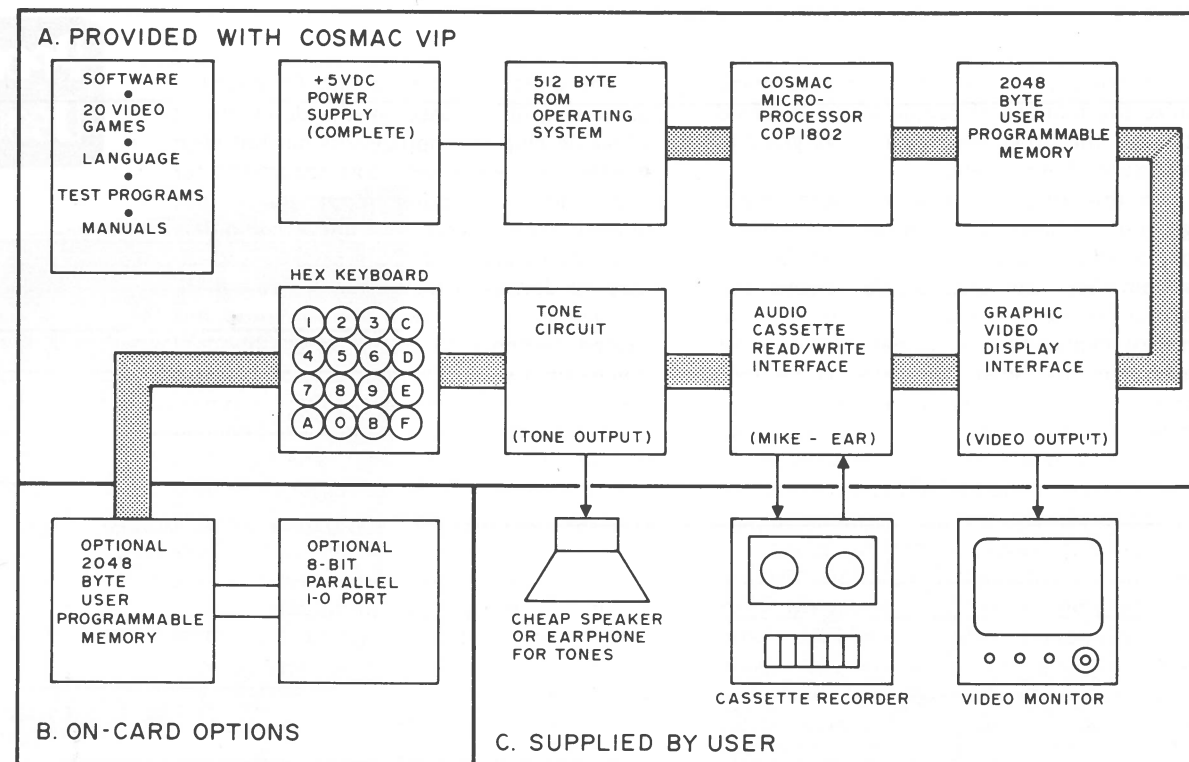


Figure 1: Block diagram depicting the essentials of a fully functioning COSMAC microprocessor. The system provides the user with everything needed to start programming except a video monitor and a cassette recorder.

computer purchase is the ability to program the computer. Suppose you buy an expensive system and then discover that you either can't or don't enjoy programming it. COSMAC VIP provides enough ready to use programs to justify the price even if you never write a program. It also provides a simple interpretive user language that many have found just as easy to learn as BASIC. This new language requires much less memory and is oriented toward simple graphics.

The fourth obstacle preventing many potential home users from getting started is the fear of buying an expensive computer and watching it become obsolete within several months. I feel that use of state of the art integrated circuits for COSMAC VIP helps minimize this problem, as does the relatively low cost of the system.

Hardware

Figure 1 shows a block diagram of the COSMAC VIP system. It shows exactly what is provided in the price and what the user has to provide. What you don't get is a video monitor, inexpensive speaker, standard audio cassette recorder and expensive cabinet. What you do get is everything else needed to immediately load and run 20 video games and write and debug your own graphic, game or educational programs. This includes all the memory you need and

an easy to use numerically oriented language. You also have the ability to use COSMAC machine code.

All system timing is derived from a crystal clock so there are no adjustments required. A single tone can be generated by programs. This tone can be used in games or sounded when a key is pressed. The hexadecimal keyboard is a reliable touch pad type, and is fully debounced by read only memory software.

The audio cassette interface was designed to allow saving programs on tape. You load a program into memory from the hexadecimal keyboard and then record the contents on an audio cassette. You can then load the program from the audio cassette any time you want to use it again. The cassette data transfer rate is about 100 bytes per second. You must manually start and stop the cassette recorder, unless you add your own start and stop relays. The reliability of any audio cassette data system may not be suitable for commercial work. Audio cassettes are however the best means of saving programs available for the hobby computer user working on a limited budget. With proper care in use they have proven themselves to be quite adequate. In the COSMAC VIP tape system, each byte on

tape consists of a start bit, 8 data bits and a parity bit. Parity of each byte is automatically checked when a cassette program is loaded into memory. A light and warning tone tell the user immediately if a parity error occurs. This makes cassette volume and tone adjustments possible. Another light provides a visual indication of tape data. This lets the user manually position the tape when recording several programs on the same cassette. The last byte loaded into memory from a cassette is always displayed on the TV screen.

The graphic video display consists of an array of spots on the TV screen. There are 64 spots in the horizontal direction. The number of spots in the vertical direction can be varied, under program control, from 32 to 128. The normal display is 64 spots wide by 32 spots high. This array of 2048 spots represents 256 bytes. If a bit is 1, the spot is white; if a bit is 0, the spot is black. Changing the states of the memory bits with a program creates patterns, pictures or numbers on the TV screen. It is possible to create moving patterns by shifting the bit patterns around using a program. Switching between different display maps by changing the contents of a single address register allows very quick display changes.

This graphic display uses the RCA CDP1861 integrated circuit which was designed for use with the COSMAC microprocessor. It provides the lowest cost graphic video display currently available with any microcomputer. It is limited to fairly low resolution applications. Higher resolution requires larger refresh memory capacity and much more expensive circuits. In five years of experience I have found this resolution adequate for large numbers of applications. A very inexpensive color graphics add on is being planned for this display.

Physically, the COSMAC VIP consists of a single 8.5 by 11 inch (21.6 by 27.9 cm) printed circuit card that sits on rubber feet (see photo 1). The hexadecimal keyboard occupies the lower right hand corner of the card. The power pack plugs into the wall and is connected to the card with two wires. Three shielded cables are needed for connection to the video monitor, the cassette recorder mike input and the earphone output jack. These cables are readily available at local electronics outlets.

Assembly time for a COSMAC VIP kit is two to three hours. It involves soldering the components into the card. No individual wiring is required, other than the above external connections. The assembly instructions and printed circuit card were designed with the experienced kit builder in mind. If

you have never built a kit you would be better off finding a friend with a sharp eye, steady hand and fine point soldering iron to help you out. Assembled versions will be available and a plastic cover for the card is planned. The printed circuit card can also be mounted in your own box with the keyboard, single operating switch and three LEDs remounted on the surface of the box. Only 19 integrated circuits are used in the 2 K byte standard system. A single +5 VDC supply comes with the kit and powers the computer. Only 350 mA, average current, is required with most of the current used by the NMOS programmable memory. The CDP1802 COSMAC microprocessor is the key to the low power and relatively few integrated circuits required for a system of this type.

System Expansion

The COSMAC VIP was designed as a complete, stand alone, graphic home computer. Hardware hackers will, of course, not be satisfied with the standard system and will want to add a variety of gadgets of their own. Expansion capability was therefore designed into the system. First, you can increase memory to 4 K bytes by just adding four ICs to the printed circuit card. This permits more sophisticated programs for a system of this class. A comprehensive, 44 line interface is provided that can be used to add almost anything to the system including up to 32 K bytes of programmable memory. This interface provides all the COSMAC microprocessor signals. A fair amount of technical skill is, of course, required by hackers who actually use these lines.

For novices who want to add existing IO devices such as relays, music synthesizers, printers, or an ASCII keyboard, an easy to use parallel IO port option is provided. Adding four readily available ICs to the printed circuit card provides an 8 bit output port and an 8 bit input port plus handshaking signals on 22 interface pads. A standard 22 pin card socket can be used for connecting external devices to these ports. The output lines can drive two TTL loads. The input lines are high impedance, 22 K ohms or more.

Software and Documentation

Two manuals are provided with the COSMAC VIP system. The COSMAC microprocessor user manual is for those who want to get involved with the details of machine language programming. The COSMAC VIP manual provides all the information required to assemble, run, program and troubleshoot the system. Every

program provided in this manual can be run on the standard 2 K byte memory system. Detailed, drafted logic diagrams are provided and a section of the manual is devoted to troubleshooting techniques.

The most important feature of the manual is the number of ready to use program listings provided. I am one of those people who find it hard to understand anything unless I can see some examples. I suspect that a lot of people are in the same boat but don't like to admit it. For this reason, plenty of actual programs are provided. You can enter these programs using the hexadecimal keyboard and record your own cassette library. These programs were developed and debugged using the COSMAC VIP system itself to thoroughly test the operating system. Most of the programs were written in the interpretive language provided with the COSMAC VIP system.

Several test programs are provided to help you identify and analyze hardware problems should they occur. A short test program that can be loaded from the hexadecimal keyboard allows recording of a test cassette for analyzing tape problems. A special memory test program is provided that will detect most types of memory bit failures and will inform you which circuit to replace.

20 video game programs are provided. These games include target shooting, number guessing, picture or pattern generation, shooting stars, etc. Let's be honest, a primary application for home computers is games. There is nothing to be ashamed of if that's all you do with your computer. Chess, Go, bridge, tennis and baseball players do not constantly apologize for playing games. The sales of commercial games and the newer video games show that games fill a real need. Many people spend most of their spare time watching people getting paid to play games on TV. Games are fun, educational and satisfy basic human needs. The computer is a super toy. Why not relax and enjoy it? The COSMAC VIP game set permits the whole family to enjoy it with you. Starting with games may even develop latent programming interests that solving equations won't.

Designing Programs

This is where the frustration starts for many beginners. A number of high level languages have been developed to make computers easier to program. These are invariably English language oriented. This is nice in theory but uses up a lot of memory in practice. The more popular languages are also aimed at math and data processing applications rather than control, games

and video graphics. A unique numerically oriented interpretive language is provided with the COSMAC VIP.

This language looks a lot like machine language. It provides 31 elementary instructions. Each instruction is two bytes or four hexadecimal digits. Single instructions permit generating a random byte, reading in a hexadecimal keyboard digit, displaying a pattern on the TV screen, sounding a tone, incrementing a variable, setting or testing a real time clock, etc. Sixteen 1 byte variables are provided. Subroutine nesting and machine language inserts are permitted. The interpreter for this language only occupies 512 bytes of programmable memory. Programs written in this language are extremely compact. A programmed video kaleidoscope requires less than 64 instructions. A rocket launching space ship intercept video game only requires 104 instructions including on screen decimal scoring. It is possible to play tic-tac-toe against the computer using a 235 instruction program that includes random mistakes by the computer. There is space for programs containing up to 592 instructions in the standard 2 K byte system.

I've been using this type of hexadecimal interpretive language programming for a number of years. It has been learned and used by children, engineers and programmers with no difficulty. An occasional professional programmer may grumble about the lack of self-documentation and the need for patches, but the money that these features cost isn't coming out of his pocket. Newcomers to computers might even find this COSMAC VIP language somewhat easier to understand than conventional high level languages. This is particularly true if they come from a hardware background. It is certainly cheaper to implement this type of interpretive language. Ease of understanding is difficult to measure since everyone is different. What's easy for one person may be difficult for another. Newcomers may welcome a programming approach that simplifies programming without straining their memory budgets. The home computer field is still young enough to tolerate experimenting with a variety of languages.

Some home computer users will want to get involved with machine language programming. They may be interested in developing their own languages or learning microprocessors at the detail level to enhance their careers. The COSMAC VIP can be directly programmed in machine code from the hexadecimal keyboard, and machine language programs can be saved on cassettes. The operating system permits examination of the

processor's registers to facilitate debugging machine code programs. The external interface permits substituting a user's read only memory based operating system for the standard one, if ever required.

Summary

COSMAC VIP is a complete computer on one printed circuit card. It is oriented toward control and video graphics. In price and performance it fits in between low priced machine language trainers and more expensive high level language oriented systems. It is ideally suited to

A practical, low-cost home/school microcomputer system

J. Weisbecker

The mention of low-cost computers usually evokes one of two images. Some see a super calculator while others picture a large-data-base processor. However, a more modest machine has been developed that could sell for under \$500 in the relatively near future. Prototypes of this low-cost, mass-market, free-standing computer system based on the RCA COSMAC microprocessor architecture have been constructed, programmed, and operated in a home environment over the past several years.

WHILE stored-program computers have been one of man's greatest achievements, the majority of people are denied direct contact with these fascinating machines. Why can't thousands enjoy computers as a rewarding new hobby? Why aren't children who may be tomorrow's social problems having their minds turned on constructively by playing with computers? Why aren't computers used more widely in the area of correcting learning disabilities? Why can't the unique recreational and educational aspects of computers be made available to everyone? Cost is the single answer to these questions. There is no shortage of ideas for using computers, but there are no computers with a mass-market price tag.

For widespread home and school use, the price of a free-standing, self-contained computer system should be well under \$500. This is the price level for color tv's, quality audio systems, home-study courses, air hockey games, pool tables, one-week vacations, cheap electronic organs, and encyclopedias.

Does the advent of LSI microprocessor and memory chips signal the availability

of low-cost home/school computers in the near future? If we need conventional input, output, and bulk storage devices, the answer is *no*. If our applications are playing chess, printing pages of data, or accessing large on-line digital/video data bases, the answer is also *no*. If we take a more modest applications approach and place reasonable limitations on hardware capability, the answer becomes a resounding *yes*. In fact, prototypes of one such system have been constructed and in use for several years.

This system is called FRED (flexible recreational and educational device) and is based on the RCA COSMAC microprocessor. The COSMAC architecture is ideally suited to this application and COS/MOS circuitry minimizes power-supply cost. Assuming the availability of 4 x 1024-bit random access memory (RAM) chips and a single-chip microprocessor, a complete system could be built using as few as ten chips. In large-volume production, a selling price under \$500 would be achieved easily. The success of products such as calculators, Odyssey, and coin-operated games indicates that the type of home/school computer described herein could achieve wide market acceptance.

personal use at home and educational use in schools.

COSMAC VIP is being made available by RCA. It is the result of five years of development effort in my laboratory which was aimed at producing a cost effective hobby computer. A low level interpretive programming language, test programs and 20 video game programs are provided with COSMAC VIP as it is coming to market for the first time. Assembled and covered up versions are planned. Add ons such as color graphics are being developed. Family fun, low cost, and flexibility were primary design goals, which I believe we've accomplished with this product.

Along with a description of the computer and uses of the prototype system, system philosophy is covered in detail since it is a prime factor in achieving low cost.

Application and system overview

In schools, FRED provides a powerful educational tool. It can be used to drill and test students from first grade up. It can be used in educational games, simulation exercises, and reading readiness. It can also be used to teach programming or as an adjunct to math courses, and as an accessible student tool in almost any subject. It lends itself to demonstrations and experiments in a wide variety of areas, such as the area of learning disabilities or for stimulating the development of creative abilities. Cost per student hour is measured in pennies.

In the home, FRED offers an extension of the school uses. It also functions as a sophisticated entertainment center for the whole family.

The low cost of the system is achieved via a combination of techniques. The use of the RCA COSMAC microprocessor is fundamental to minimizing total system cost. A basic memory of only 1,024 bytes keeps RAM cost low.

Since FRED is a stored-program computer, it requires a program to be loaded into memory before use. Program loading is performed with an inexpensive audio cassette player which also gives the computer voice, music, and sound-effect capability.

After being loaded with a program cassette, FRED is operated with a small 16-position keyboard. For a game, the player would press appropriate keys to

indicate the moves. Overlay cards are provided so that keyboard labeling can be changed for different programs.

FRED is attached to the antenna terminals of any tv set. This provides an inexpensive, flexible, dynamic output display which is ideally suited to home/school use. Numbers, words, or simple pictures can be displayed on the tv screen in the form of dot patterns.

Adding a \$25 punched-card reader and \$10 manual punch to the basic system enhances its usefulness and provides more sophisticated users with the ability to prepare and save short parameter lists or programs. Adding a module for recording the contents of memory on cassettes converts the basic FRED system into a user-programmable computer for serious hobbyists. Other possible attachments include light guns, extra memory (RAM), pre-stored programs or tables (ROM), and output relays for control uses.

Design considerations

Two different approaches can be taken toward developing an under \$500 home/school computer. One approach involves specifying a desired set of system characteristics and then attempting to achieve cost objectives. The danger in this approach is the tendency to overspecify the hardware so that price targets cannot be met. The approach described here involved defining a minimum-cost, non-trivial hardware system that could easily meet a low-price goal, and then testing its usefulness. This approach ensures that applications development effort will not be wasted.

Any free-standing computer must include the following:

- 1) Central processing unit (CPU)
- 2) Main memory (RAM)
- 3) Input device(s)
- 4) Output device(s)
- 5) Bulk storage device(s)

The FRED system required the development of a philosophy that was consistent with utilizing minimum cost hardware. This philosophy included asking what we could do with inexpensive devices instead

of asking what types of devices we might ideally want. Because of our low-price goal, the system implications of each device choice were magnified in importance.

The area of use is also a consideration involved in developing a minimum-cost system. For mass-market recreation/education use, competitive cost-performance ratios can be largely ignored. Reliability can also be sacrificed to some extent to achieve low cost. This does not mean that permanent system failures can be tolerated but that occasional transient errors during program loading will not be catastrophic. The need for memory and processor parity checking was also felt to be an unaffordable luxury.

Ease of use is a primary requirement. Initial users will be completely naive and frightened off by any apparent complexity. Thus, it was decided that a turn-key system philosophy would be adopted. The user merely loads a program from a library to obtain a desired game or function. He is not expected to program the machine. This approach eliminates the need for program dump/save capability and maximizes ease of use. Also, the need for an expensive control and diagnostic panel is eliminated. In fact, only two switches are required for basic use — LOAD and RUN.

A block diagram of the basic system is shown in Fig. 1. System considerations will be included in the discussion of individual elements.

Central processing unit and memory

The advent of single-chip LSI microprocessors makes the consideration of under \$500 systems possible. Suitable microprocessor chips should be available at less than \$25 within the next several years. The choice of a microprocessor has

a large influence on total system chip count and cost. This influence is an important consideration since the microprocessor itself is only a small part of a total system cost. For this reason the home/school computer described here was based on the RCA COSMAC microprocessor architecture.

This architecture immediately eliminates the need for a read-only memory (ROM) in the minimum system. Only one supply voltage is required, and COS/MOS circuits further reduce system power-supply costs. A self-contained direct-memory access (DMA) channel facilitates initial program loading and display refresh. A single-phase clock is consistent with minimum cost. High output drive capability eliminates external buffer circuits.

The 8-bit COSMAC architecture is compatible with the intended uses of the system. The short, single-byte instruction format permits compact programs leading to minimum memory requirements. Since the average user will never see the processor micro-instruction set, ease of programming is secondary to efficient memory utilization.

For the hobbyist, COSMAC provides a simple, easy-to-understand set of micro-instructions. This means that he will not be confused by the subtleties of a complex order code. He can, instead, concentrate on his applications.

A complete description of the COSMAC microprocessor has appeared previously^{1,2} and will not be repeated here. This architecture has demonstrated its advantages in prototypes of the low-cost home/school system.

Because of the nature of our application, RAM is required for both program and data storage. It is well known that programs tend to expand to fill available memory space. Providing a 4096-byte

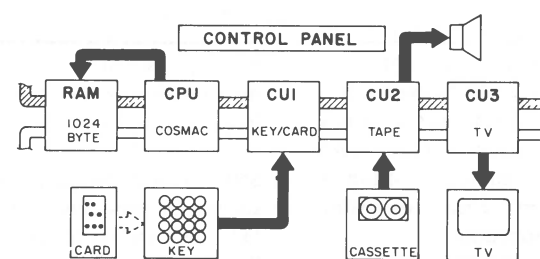


Fig. 1 — Basic system.

memory only ensures that no program will be written requiring a smaller memory. Even projecting a cost of \$0.02/byte would yield a cost of \$82 for a 4096-byte memory. This size memory would add \$200 or more to the selling price of the system. Instead of asking how much RAM we could use, we provide 1024 bytes in the minimum system. This is consistent with keeping memory cost equal to projected microprocessor chip cost. Should LSI memory costs drop below \$0.02/byte we can increase minimum system capacity to 2048 bytes or lower the price of the 1024-byte system. Based on current trends, we can safely predict one-microsecond LSI RAM costs of \$0.02-\$0.03/byte. Dynamic RAM chips are at this cost level now, while static, single-voltage RAM chips are currently available at \$0.07 to \$0.08/byte.

The challenge of a 1,024-byte memory seems to stimulate cleverness in programming and makes a future 2,048-byte memory seem large by comparison. If we had initially provided a 4,096-byte memory, subsequent size reduction to meet cost targets would have been extremely difficult. The probable availability of 4 x 1024-bit RAM chips within the next several years should result in a minimum system requiring only two chips for memory.

Limiting the minimum system memory to 1024 bytes also provides several system cost advantages. Power-supply cost is reduced, memory-address drivers are eliminated, and printed-circuit board space is saved. A less obvious system implication is the effect of memory size on program loading costs.

In general, the user should be able to load a program in half a minute or so. This coincides with observed user patience factors. An occasional error requiring reload can also be tolerated for short load times. This permits lower reliability loading devices to be used. To load a 1024-byte memory in 30 seconds requires a serial transfer rate of only 300 bits/s. This assumes a parity bit for each byte. For a 4096-byte memory, the required rate jumps to 1200 bits/s. The required transfer rate influences the choice of a program loading technique. Lower rates can generally be translated into lower costs and better reliability.

It is in the area of input, output, and bulk storage that we encounter the major cost

problems. The choice of I/O and bulk-storage techniques also has a major effect on the range of possible system applications.

Output display

Fortunately, an ideal, low-cost output device for home/school applications already exists. A standard tv set provides a flexible, dynamic output display device that most users already own. Even if a tv set must be purchased for \$50 to \$100, this cost can be charged largely to normal viewing use.

The choice of a tv display format involves a number of system considerations. These include types of applications, display-refresh memory requirements, and complexity of control circuits. A low-resolution, black-and-white dot matrix was chosen for maximum flexibility at minimum cost. In this system, an array of white dots is displayed on a black background. The black background avoids potential picture noise problems. Arrays of 32 x 32, 16 x 64, and 32 x 64 dots are provided. Fig. 2 illustrates the flexibility of this format for displaying small game boards, simple pictures, words, numbers, or symbols. Each dot represents the state of a main memory bit. If the bit is "1" the dot is on, if the bit is "0" the dot is off.



Fig. 2 — Display flexibility.

Simple animation can be achieved by modifying memory bit patterns at appropriate time intervals. Any contiguous 128/256-byte section of memory can be selected for display by setting a microprocessor address pointer. This display pointer can be modified at any point in a program providing the ability to step through various memory display areas at any desired rate. It is easy to flash selected portions of a picture by alternating between two display areas in memory.

For 32 x 32 and 16 x 64 displays only 1024

bits (128 bytes) of memory are required for display refresh. This is only 12.5% of the minimum system memory but provides a larger-area picture when required. It is also useful in expanded memory systems. It should be emphasized that no ROM is required for tv display in the minimum system and that frame refresh storage is provided via main memory.

The tv control unit (CU3 in Fig. 1) contains the circuits for generating tv sync signals and for requesting memory bytes via the COSMAC DMA channel as required for display refresh. The individual bits of each byte are used to generate a video signal. The composite sync and video signal modulates the output of a simple rf oscillator. This modulated rf signal can be applied to the antenna terminals of any standard tv set.

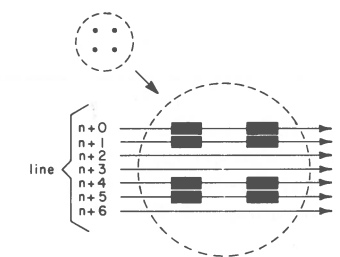


Fig. 3 — TV dot detail.

Fig. 3 illustrates the detailed timing for displaying dots on the tv screen. A magnified view of four dots is shown. Each dot is two horizontal tv lines high with a two-line space between dots. An 8-byte row buffer is provided in the tv control unit. Each tv line time is 65 ns. During the two blank line times, between rows of dots, up to 8 bytes (64 bits) are retrieved from main memory and stored in the row buffer. During the next two tv line times, the bits in the row buffer modulate the tv beam to display the proper dot row pattern. By spreading the dots as shown, the low-resolution display fills up the tv screen, and memory to tv refresh transfer rate is lowered.

The tv control unit also generates a program interrupt signal at the beginning of each tv frame. This interrupt permits the program to initialize the microprocessor display address pointer at the appropriate time. Since tv program interrupt occurs 60 times/s, a free real-time clock exists when needed. This clock

capability is useful for timing purposes in a number of applications.

Bulk storage

Program library storage and loading present another major problem area in a low-cost system. The high cost of existing computer devices such as floppy discs and digital tape units immediately rules out their use. Paper tape is awkward and still fairly expensive. Conventional punched-card readers are expensive and inconvenient.

This problem was solved by using another existing, inexpensive consumer device — the audio cassette recorder. Suitable portable units sell for under forty dollars. A built-in unit could be provided for less than \$20. Since normal use is not impaired, the cassette recorder cost is spread over normal and computer uses. Several methods for storing bit serial digital data on audio cassettes have been described,^{3,4} and others are possible. We developed a proprietary, pulse-counting technique that yields a 50-byte-per-second transfer rate, tolerates missing or extra pulses, and permits tape-speed variations of 30%. This system works well even for cheap portable audio units. To minimize costs, only single-track capability is required in the system.

Since errors can be expected every so often, a parity bit is added to each byte on tape. The cassette control unit checks the parity of input data read from tape and turns on an error light for incorrect parity. Reloading a program when an error occurs is a simple, quick procedure.

Fig. 4 shows the single-track, cassette tape format used. Digital or audio blocks are always framed by 4-Hz stop tones (T). The stop-tone detection circuit is designed to respond only to long (0.5 s) continuous tones so that voice or music frames will not falsely trigger it.

Fig. 5 shows how a standard cassette player is used in the system. Most cassette

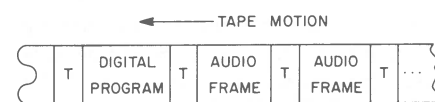


Fig. 4 — Cassette tape format.

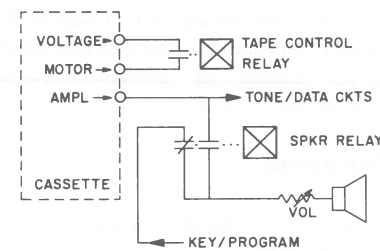


Fig. 5 — Cassette attachment.

recorders provide an external speaker or earphone output jack. This output is connected to the control unit as shown. Stop tones and digital data are detected via this cassette output line. A relay is also provided that permits the cassette output to be connected to a speaker under program control. This allows selected tape frames to be passed inaudibly.

The majority of inexpensive cassette recorders have a remote start-stop control jack. This is designed for use with a microphone or foot switch. For use in our system the cassette remote jack is connected to a program-controlled relay. This gives the computer the ability to start and stop tape, providing the user has previously placed the cassette recorder in its PLAY mode.

The primary-system operating controls comprise two toggle switches ... LOAD and RUN. The LOAD switch activates the cassette control unit (CU2 in Fig. 1). The desired program cassette is selected by the user, rewound, and the recorder set to PLAY. When the first stop tone is encountered the data-reading circuits are automatically turned on. Waiting for this stop tone eliminates possible noise problems at the beginning of the tape. The digital data representing the program is loaded sequentially into memory at 50 bytes/s. The second stop tone automatically stops the tape via the tape-control relay. Turning off the LOAD switch resets the computer. The RUN switch initiates execution of the program just loaded.

During program execution the tape can be automatically restarted so that the user will hear audio frame No. 1 at a desired time. The stop tone following audio frame No. 1 will automatically stop the tape. The program can monitor the state of the control relay to determine when the end of data/audio frames occur. This permits synchronizing audio material on cassettes with a program.

The provision for program-controlled audio segments has an important system implication. The ability to provide instructions, questions, or other data in the form of voice frames on tape minimizes the need for a high-resolution, alphanumeric tv display with its attendant requirement for large refresh and backup digital storage capacity.

The speaker provided for use with the cassette recorder provides a useful output device by itself. A flip-flop that can be set and reset by program drives the speaker when it is disconnected from the cassette output. Programs can, therefore, create any audible sequence of tones desired.

Input devices

The primary input device for our system is a 16-position keyboard. A number of \$5 to \$10 keyboards of this type have been developed for use in pocket calculators. A flat, printed-circuit type was chosen to facilitate an overlay feature. A slight modification of the keyboard permits insertion of a printed card above the switch array. Various cards are provided to relabel the switch array for different programs. Some programs require symbol keys, others letters or numbers. For educational programs, keys can be labeled with colors, pictures, words, or possible answers to questions. For other programs keys might be labeled with colors, pictures, words, or possible answers to questions. For other programs keys might be labeled with direction arrows for manipulation of the tv display. The variable-label keyboard is fundamental to meeting the ease-of-use criterion for this type of system.

Unfortunately, the flat keyboard, which is ideal for variable labeling, has no tactile feel. This objection was overcome by taking a systems approach. Since a speaker already exists, switch depressions need only be coupled into this speaker to provide an audible "click". This has proven to be an adequate substitute for tactile feel. The scanning approach used to decode the switch panel minimizes the cost of this approach. Specific programs can also generate various tones for switch depressions, which again substitute for tactile feel.

The 16-position keyboard normally causes an 8-bit byte to be stored in memory for each key depression. The

most significant four bits (digit) are normally 0000. A shift switch pressed in conjunction with a hex key causes the most significant four bits to be 0001. The least significant four bits of a stored byte represents the code for one of the 16 possible hex digits shown in Fig. 6. The hex keyboard in conjunction with a shift switch permits entry of 32 different byte codes.

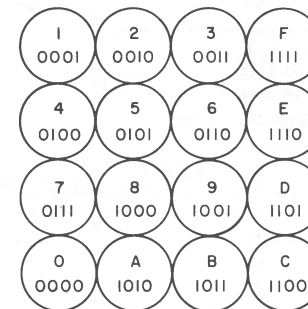


Fig. 6 — Keyboard code.

An alternate mode of keyboard entry is also provided. In this mode two key depressions per byte are required. The first key specifies the most significant hex digit of the byte to be entered. The second key provides the least significant hex digit of the byte. This mode provides the sophisticated user with a convenient way to manually load his own machine-language programs. It is also a useful mode for certain turnkey programs.

The hex keyboard control unit (CU1 in Fig. 10) also supports the addition of an inexpensive card reader to the minimum system. This unique device uses 3-x 5-in. punched cards. Data is punched in the form of rows of holes. Fig. 7 shows four such rows (A,B,C,D) punched on one side of a card (both sides can be punched). Each row represents the 4-bit code for one hex digit, plus a parity bit to make the 5-hole code odd. This means that at least one hole will be punched for each of the possible hex digit codes. Cards are read by dropping them into a 3-in. slot. They fall past a light source and six inexpensive photodiodes. One photodiode senses the

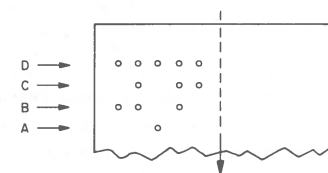


Fig. 7 — Punched card.

presence of the card and conditions the control-unit circuits accordingly. The other five photodiodes read the self-clocking hex digit codes into the system. Hex digits are paired to form bytes before storage in memory.

By limiting the information content of a card to 16 hex digits per side, the mechanical tolerances of the reader are not critical. The reader has no moving parts and photodiodes can drive the COS/MOS control unit circuits directly, eliminating the need for amplifiers. These factors combine to provide a very low-cost input device (\$25 or less).

The low-cost card reader can be used to enter short lists of parameters or short user-prepared programs. In a classroom the teacher might use parameter cards to set up the difficulty of test/drill programs. Picture cards used by the student could contain the spelling of the word pictured for checking by the computer. The cards also facilitate certain simulation languages and permit users to save simulation language programs that they develop themselves.

Another low-cost, optional input device is a simple light gun. This device contains a lens system and a photodiode. The computer detects when the gun is pointed at any lighted area of the tv screen. The light gun facilitates various computerized target-shooting games. By alternately flashing portions of the tv display a program can determine the area at which the light gun is pointed. This permits the user to indicate various types of choices by pointing the gun at appropriate portions of the display.

Applications philosophy

The open-ended aspect of a stored-program computer differentiates it from other types of recreational and educational devices. Any number of special-purpose devices such as tv games, shuffleboard tables, electric football games, and educational toys are ideally suited to their intended function. None of these, however, will change their characteristics as user moods or interests change. Many of these special-purpose devices are seldom used after their initial novelty wears off. The stored-program computer is a dynamic general-purpose device. New programs constantly adapt it to changing moods and interests without

the expense of new hardware.

The real value of the home/school system described here lies in its ability to stimulate and develop human capabilities that are often ignored or discouraged by conventional recreational and educational device approaches. The computer system provides an environment that stimulates experimentation, analysis, and creativity. Contemporary tv encourages passive viewing. With a computer attached to his tv set the user is encouraged to interact with the tv picture to play games.

For a child, the computer may initially provide arithmetic or spelling drills. Even his memory development can be made more interesting via interaction with the computer. However, the child will eventually begin to wonder about the computer. Programs made available stimulate his curiosity and let him experiment with changing game rules. He can even begin to formulate and develop his own simple programs in a variety of simulation languages. While the initial use of the computer involves memory skills, it eventually encourages experimentation and the development of analysis and synthesis capabilities.

The creation of programs that stimulate the user to develop mentally is a challenging task with a high payoff in terms of satisfaction. We have only begun to explore this area of use for very small, inexpensive, practical computers of the type described here. Even so, the number and richness of uses for this type of system are surprising. After experience with 64,000-byte main memories and large disc files, we are apt to dismiss a 1024-byte memory system as unusable. To dispel this notion, over 80 specific applications of the inexpensive home/school system are listed in Table I. Many of these represent whole classes of programs that could be developed. Types of programs already written are marked with an asterisk.

Four general areas of use are identified in Fig. 8. These areas are discussed individually although there is a high degree of overlap between them. Most of the listed uses require only the basic system. Ref. 18 also describes a number of uses (mostly games) that have been programmed on larger computers with hard-copy output. Many of these are readily adapted to the low-cost computer described here.

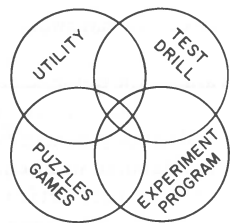


Fig. 8 — Areas of use.

Utility applications

This area involves using the computer to achieve some specialized function. These functions might include those listed under "utility applications" in Table I.

A simulated four-function decimal calculator has been implemented on the basic 1024-byte memory system. This capability includes display refresh, digit pattern tables, decimal arithmetic algorithms and 20 digit operand/result capability. A 2048-byte memory would permit a programmable calculator with multi-line display. Optional ROM chips could provide a permanently resident calculator capability if desired.

A variety of specialized calculators can be implemented on the basic system. Programs to provide scorekeeping for card, war, or other commercial games could be provided. Children could have their own secret-code computer. For several years a plastic toy rock-identification computer has been on the market. With this equipment, certain tests are performed (color, hardness, etc.) on a mineral sample. The plastic computer and a set of cards is then used to identify the sample. The basic home/school system could readily be programmed as a classification computer of this type.

Logic machines have held a certain fascination for years.⁵ The computer readily simulates a variety of machines of this type. It can also be programmed to simulate gambling algorithms. A pair of dice is easily simulated for use in a number of games. Random-number generating machines find use in various school courses and experiments. Serious war-game fans can use computer-generated battle results and score-keeping to advantage. The leading magazine in the field, *Strategy & Tactics*, has over 20,000 subscribers, indicating a wide interest in this type of activity.

Table I — Typical applications for home/school microprocessor.

Utility

Four-function decimal calculator*
Hex/binary calculator
Game score keeper
Number base converter*
Weight/measure converter (metric)
Secret code computer
Logic machine (see Ref. 5)
Classification computer
Gambling strategy computer
Other specialized calculations
(temp. conversion, interest, etc.)
Electronic dice
Random-number generator
Simulation game computer
Bar graph
Interactive audio-visual toy
TV greeting card*
Electronic "etch a sketch"*
TV puppet
Audio-visual demonstrator*
Mind-reading computer
Party compatibility computer
Programmed timer/controller
Stop watch/game timer
Simple electronic organ
Metronome
Advertising display

Text & drill

TV arithmetic drill*
Word spelling drill*
Word recognition test*
Pattern recognition (superimposed, complex)*
Electronic flash cards
Classroom group games
Preschool shape/color recognition
Up-down, left-right discrimination
Sound-picture matching
Reading readiness skill drills
Logical aptitude test (see Ref. 6)
Number base conversion drill*
Flap board simulator (see Ref. 7)
Morse code drill
Reflex testing
Logical deduction test (21 questions)*
Logidex (see Ref. 8)
Memory training (sobriety test)
Individual testing & scoring aid

Change making drill
X-Y curve plotting drill
Time sense development

Games & puzzles

TIC TAC TOE*
Hexapawn (see Ref. 9)*
Sliding block puzzles*
State change games/puzzles (see Ref. 10)*
Bowling*
Football (see Ref. 11)
Minikrieg*
Target shoot (optional gun)*
One-armed bandit*
Network games*
Twenty-one*
Cell matching games*
Maze tracing (invisible, changing)*
Race games (against time)*
Space war*
Bombs away
Combinational/sequential puzzles (see Ref. 12)
Dodge games (space ship & asteroids)
Fish card game
Moon landing
NIM games (static/dynamic)*
Invisible counter board games
Simulation games (see Ref. 13)
Game, forms of utility/test/drill programs

Experimentation and Programming

LIFE (see Ref. 14)*
Penny Matching computer (see Ref. 15)
Turing machine (see Ref. 16)
Tutorial computer*
Picture computer
Sound computer
Machine code programming
Simulations
Variable rule games
Logic simulator
Learning machines
Probability & Monte Carlo experiments
Heuristic program design

For very young children the computer simulates a variety of interactive, audio-visual toys that make sounds and change tv pictures in response to key depressions. Customized and animated TV greeting cards/decorations for birthdays, Christmas, or Halloween can be provided. Simple, key-operated tv puppets are possible. Also, the ability to step a spot around the screen permits drawing tv pictures.

The ability to synchronize audio-tape frames with programs permits programmed audio-visual tutorials for

home and school or eye-catching advertising displays. The basic system realtime clock facilitates key-operated game-timer or stop-watch capability. The program-controlled speaker turns the computer into a simple electronic organ or metronome. TV display can be included with the sound generation.

The basic system can be used for any number of party games including couple compatibility testing. Adding inexpensive, output relays lets you program Christmas tree light sequencing or control other devices.

Test and drill applications

Drills mainly involve the development of memory or conditioned reflexes. Testing can involve a wider range of skills. The infinite patience of a computer makes it ideal for drills. Interactive capability adds interest and motivation. Some specific examples are shown in Table I.

Programs of this type are ideal for individual use to overcome specific weaknesses. The psychological benefit is that the computer does not seem to be making value judgments of a child during a drill as a child might fear a teacher or parent would. The drills can also be made to appear as games with the computer providing added motivation and self-adjusting challenge levels.



Fig. 9 — Add drill display.

A simple arithmetic drill might appear as shown in Fig. 9. First, addition problems are randomly generated on the tv screen. Next, the child must enter correct answers via the keyboard in time to prevent the boat from completely sinking. The rate at which the boat sinks can be preset by the teacher and changed from session to session to maintain challenge as the child's speed and accuracy improves. The computer displays the child's score when the teacher enters a special code (key/card).

Spelling drills can be implemented in several ways. A cassette voice could ask for the spelling of a word which the student then spells via the keyboard. The tv display and/or audio tone tells him if he is right. The computer again keeps score and times answers. A simple word-recognition drill involves displaying a word on the tv screen and asking for the corresponding picture via keyboard or card input. Patterns can be superimposed on the tv screen and the student asked to identify the components of the picture. Simple preschool shape, color, or sound-recognition programs are possible. Up-down, left-right concepts are readily presented via tape voice and animated tv displays.

The computer can be programmed to momentarily flash a picture, word, pattern or group of symbols on the tv screen to develop perception skills.

Reflexes or time sense can be developed by requiring a specific keyboard response following programmed sounds or tv displays. Scoring adds a motivating game element to these types of drills. Morse code is taught by requiring the translation of tape voice passages into key depressions. The computer checks accuracy and gradually increases speed.

Reading-readiness skills include simple shape recognition, word configuration recognition, and maintaining fixation on a moving object. The latter could involve having a child press direction-changing keys to prevent a moving spot on the tv from hitting obstacles.

The computer can easily simulate logical aptitude testing devices⁶, existing simple educational aids⁷, or games.⁸ The dot-array tv display format is ideal for X-Y plotting practice. The computer can be used for individual testing and scoring in any subject area and at any grade level. The test questions are provided in printed page or booklet form. The computer specifies which questions are to be answered via tape voice or TV display. Answers can be in the form of multiple choice, numbers, or words which the computer can check against a pre-stored table of correct answers.

The ability to skip audio frames on tape via the program-controlled speaker relay provides added flexibility for test and drill applications. Two sequential voice frames could be provided per question. One frame would "tell" him that his answer was wrong, and why. For each student response the computer "plays" the appropriate frame and inaudibly "skips" the other.

Games and puzzles

Games and puzzles are normally associated with recreation. We have already seen that a number of utility programs have recreational aspects. The educational as well as recreational aspects of games and puzzles are discussed here. Some of the possible uses of the computer in this area are listed in Table I.

One of the most obvious aspects of the list in Table I is that there has been no problem in motivating people to write game programs. TIC TAC TOE, Hexapawn, Twenty-one, and space war are played against the computer. Two-player versions are, of course, possible.

Hexapawn⁹ was implemented as a learning program. The computer learns to play perfectly only after a number of games have been played. This type of learning program provides the basis for experiments where the user plots games played versus games lost by the computer to establish empirical learning curves.

Bowling displays the pins on the tv screen. A ball spot randomly moves up and down at the opposite end of the alley. Pressing a key at the proper time rolls the ball. Sound effects, scorekeeping, and some random factors are incorporated in this two-player game. A variety of football games are possible. The simplest of these involves simulating commercial varieties of electrical football.¹¹ Minikrieg is a simplified war game. With the optional light gun a variety of computer-controlled target-shooting games can be devised.

Sliding block puzzles are easily simulated via the tv display. A move counter keeps the puzzle solver honest. Cell matching games involve momentarily flashing an array of cells on the tv screen at the beginning of the game. Each cell contains a symbol. Two players take turns trying to find cell pairs with matching symbols. The most matches wins the game. Network games involve completing a path from one point to another before your opponent. A number of published combinational/sequential puzzles can be easily simulated.¹²

Manipulating a moving spot through a racecourse or maze in the minimum time has proved to be a popular pastime. Spot acceleration and deceleration maximize the challenge. The computer also permits invisible and changing mazes to be easily implemented. NIM type games can have a dynamic aspect included by using the computer. Board games can incorporate invisible counters whose positions must be deduced. Moon landing involves selecting fuel-burning rates so as to avoid crashing. This type of game lends itself to experimentation, since it represents a simple simulation situation.

Experimentation and programming uses

This area might be thought of as primarily educational. There is, however, a recreational aspect as well. Developing programs for your own computer embodies both educational and recreational aspects. Some of the specific experimentation and programming

possibilities are listed in Table I.

A classic example of experimentation via computer is provided by Conway's game of LIFE described in Ref. 14. The hours of bootlegged computer time devoted to this program at computer centers all over the country are a testament to its recreational value. LIFE simulates a succession of generations for a colony of cells. Cell birth, survival, and death are controlled by algorithms in the program. Watching the patterns of cells change for each generation on the tv screen is addicting. The program is extremely rich in experimental possibilities. New starting patterns that yield interesting and sometimes surprising life histories are constantly being discovered.

Heuristic programs for simple games such as Hexapawn and Penny Matching^{9, 15} let the user develop experimental learning curves. This approach has been used to add interest to grade-school math even without the availability of a computer.¹⁷ Letting the user modify program behavior via keyboard parameters stimulates more creative and sophisticated experimentation; even TIC TAC TOE becomes a fascinating educational device when the user is allowed to modify the rules that the computer uses to play.

The computer can provide a variety of simple simulations that encourage experimentation. A simple moon-landing game or racecourse game with acceleration and deceleration controls are examples. A logic simulator would permit inexpensive experimentation with arrays of logic elements especially, since commercial hardware logic trainers are quite expensive. A random-number generating program facilitates experimental development and understanding of probability curves. Game-theory experiments are a natural application. Yet, none of these uses requires programming ability on the part of the user.

However, the area of programming provides the richest and most valuable recreational and educational experiences. Programming capability can be provided at several levels. A simple set of simulated instructions to move a spot around the tv screen could be provided via card or key symbols. Programming this picture-drawing computer could be introduced as early as second or third grade. The ability to program sequences of audible tones (or

music) via a simple simulation language is also easily provided.

At a slightly higher level of sophistication, various tutorial computers can be simulated. A simple fixed-word decimal computer was simulated on the basic 1024-byte system. This included ten instructions, 100 words of user memory, and a simulated control/debug panel. Teenagers were able to write and debug their own programs with as little as one hour of instruction.

Ref. 16 is of particular interest. In this article, the construction of a hardware Turing machine model for educational purposes is described. The authors list several disadvantages inherent in the alternative approach of computer simulation:

- 1) Computer time is too expensive.
- 2) Students have to learn how to operate the computer which has nothing to do with the simulation.
- 3) Graphic output display is expensive and printed output is slow and inconvenient.

The home/school system described here readily overcomes all three objections.

For the sophisticated hobbyist the area of machine-code programming will be of major interest. All that is required is the addition of circuits that permit "writing" on cassette tapes. This is an inexpensive option. The use of a small set of programming conventions together with specialized subroutines permits the sophisticated user to develop, debug, and save his own machine-code programs.

Conclusions

A practical, inexpensive, free-standing computer system based on the RCA COSMAC microprocessor would, for the first time, permit widespread access to computers. Much of the public awe and confusion relative to computers would be dispelled. The creation of a group of home computer hobbyists will stimulate invention and development of new computer devices and applications.

More importantly, educational benefits are unlimited. Computers are considered to be useful tools with which to achieve a specific end result such as processing a payroll or calculating a trajectory. This view of computers has often carried over

into educational applications with the computer cast in the role of teacher/tutor. The low-cost home/school system described here is intended as a flexible plaything which encourages experimentation and stimulates a desire to learn. This approach may be more significant than the improvement of teaching methods for unmotivated students.

Acknowledgments

A. R. Marcantonio, C. T. Wu, and B. J. Call have made a number of contributions to the development of this low-cost computer system. A. D. Robbi and P. Russo have provided continuing moral support, ideas, and major assistance in developing and evaluating the system. R. O. Winder deserves most of the credit for making the development of the system possible.

References

1. Weisbecker, J.A.: "Simplifying Microcomputer Architecture," *RCA Reprint RE 19-5-10*.
2. Swales, N. and Weisbecker, J.: "COSMAC A Microprocessor for Minimum Cost Systems," 1974 INTERCON, Session 17/2.
3. "Putting Data on an Ordinary Audio Recorder," *The Electronic Engineer* (May 1972) p. DC-9.
4. Wolf, E.: "Ratio Recording for Lower Cassette Recorder Cost," *Computer Design* (Dec. 1972) p. 76.
5. Gardner, M.: *Logic Machines and Diagrams*, McGraw-Hill, 1958.
6. Opler, A.: "Testing Programming Aptitude," *Datamation* (Oct. 1963) pp. 28-31.
7. Jones, J.: "The Flap Board: A Simple Diagnostic and Remedial Tool," *Educational Technology* (Jan. 1974) pp. 59-61.
8. Nurse, H.: "Logidex," *Popular Electronics* (Nov. 1973) pp. 63-66.
9. Hughes, J.L. and Engvold, K.J.: "Hexapawn: A Learning Demonstration," *Datamation* (Mar. 1968) pp. 67-73.
10. Schwartz, B.L.: "Mathematical Theory of Think-A-Dot," *Mathematics Magazine* (Sept.-Oct. 1967) pp. 187-193.
11. Graf, R.F. and Whalen, G.J.: "Electronic Football Lets You Play Like the Pros," *Popular Mechanics* (Oct. 1967) pp. 147-149, 228.
12. Cuccia, J.W.: "The Princeps Puzzle," *Popular Electronics* (May 1971) pp. 27-32.
13. Zuckerman, D.W. and Horn, R.E.: *The Guide to Simulations Games for Education and Training*, Information Resources, Inc. 1973.
14. Gardner, M.: "John Conway's New Solitaire Game LIFE," *Scientific American* (Oct. 1970) pp. 120-123.
15. Hagelbarger, D.W.: "Seer, A Sequence Extrapolating Robot," *IRE Transactions on Electronic Computers* (Mar. 1956) pp. 1-7.
16. Gilbert, I. and Coh, J.: "A Simple Hardware Model of a Turing Machine: Its Educational Use," *Proceedings of the ACM Annual Conference* (Aug. 1972) pp. 324-329.
17. Ackerman, J.: "Computers Teach Math," *The Arithmetic Teacher* (May 1968) pp. 467-468.
18. Ahl, D.H., Ed.: *101 Basic Computer Games*, Digital Equipment Corp., 1973.

20-MHz CMOS-on-sapphire COSMAC microprocessor

Silicon-On-Sapphire (SOS) technology has produced a high-speed version of the COSMAC microprocessor three times as fast as its bulk-silicon counterpart.

G.R. Briggs
S.J. Connor
J.O. Sinniger
R.G. Stewart

CMOS technology is well established in low-power-dissipation and high-noise-immunity applications. Implementing CMOS with Silicon-On-Sapphire (SOS) technology, however, provides additional advantages of increased speed and greater circuit density.¹ This paper describes circuit approaches developed to exploit these advantages in the design of a high-speed SOS version of the COSMAC bulk-silicon CMOS microprocessor.^{2,3}

The low parasitic capacitance of the SOS structure permits a three-fold increase in maximum clock rate—from 6.4 MHz for the bulk COSMAC microprocessor operating at 10 V to 20 MHz for the SOS design at the same voltage. The higher clock rate allows most COSMAC instructions to be fetched and executed in 0.8 μ s. Power dissipation ranges from less than 1 mW in the quiescent state to a

maximum of 100 mW when executing instructions with a 20-MHz clock. When operating at 10 V, the speed capability of the SOS microprocessor far exceeds that of other MOS devices and can be matched only by bipolar devices requiring multiple-chip CPUs and far greater static and dynamic power dissipation.

Chip architecture

The simplicity of the RCA COSMAC architecture, combined with a 5-transistor static storage cell, produces a completely static single-chip CPU with only 4,827 transistors.

Fig. 1 is a block diagram of the CPU. The principal features of the design are a 16-register scratchpad memory array, an 8-bit serial arithmetic logic unit, and extensive control logic for I/O devices. Each register of the scratchpad memory can be accessed

either as two 8-bit data bytes (RHi, RLo), or as one 16-bit external memory address. To provide the I/O control capability and remain within a 40-pin package limitation, the 16-bit address is time-multiplexed on 8 pins (MA output). Incrementing of the 16-bit address is accomplished by two passes through an 8-bit incrementer.

The scratchpad registers use a unique 5-transistor storage cell (Fig. 2) that can be laid out in an area of 10.2 mil². Access in and out of the cell is via a single n-device transmission gate, T1, and a single bit line. Two adjacent cell columns contain corresponding-order bits of the RHi and RLo bytes. The column bit lines are multiplexed to the 8-bit incrementer input and memory address (MA) latches. Placement

Final manuscript received January 25, 1977.

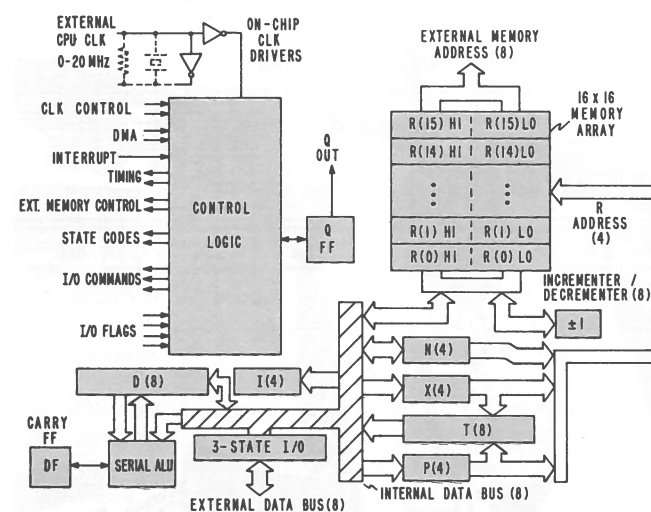


Fig. 1
SOS microprocessor, showing 16 by 16 scratchpad memory array, R, for temporary storage of data and external memory addresses; incrementer and serial ALU for altering register contents; registers N,X,P,T, and D for data manipulation; and logic for microprocessor control.

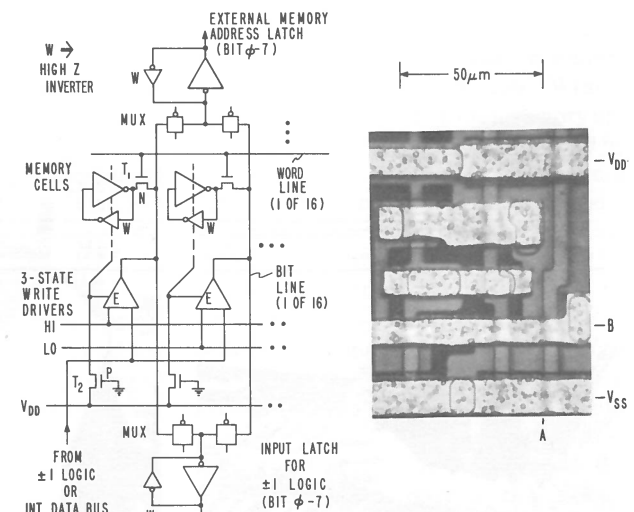


Fig. 2
Scratchpad memory (left), using the 5-transistor static storage cell (right). The supply voltages to the cell columns are reduced via transistors T2 during Write-1 operations to improve cell writing through the source-following gates T1.

of the two latches on opposite sides of the memory array permits access to the MA pins without the need for additional busing (see chip photograph, Fig. 7). Memory operation is described more completely in following sections.

Chip layout

The silicon-gate SOS process used obtains maximum circuit density and highest-speed operation for the chip. Also, a unitized logic cell layout procedure produces a logic layout optimized for minimum wasted chip area.

The wafer fabrication process used for the microprocessor includes two major improvements to the original aluminum-gate SOS process.

First, a self-aligned polysilicon gate obtained denser circuitry, increased transistor G_m , and reduced gate overlap capacitance.

George Briggs contributed to the development of high-speed silicon-on-sapphire counters and frequency synthesizers before working on the SOS microprocessor described here. He has also conducted research on a number of memory devices and systems. Dr. Briggs has received several RCA Achievement Awards and holds 22 U.S. patents.

Contact him at:
LSI Systems Design
RCA Laboratories
Princeton, N.J.
Ext. 2276

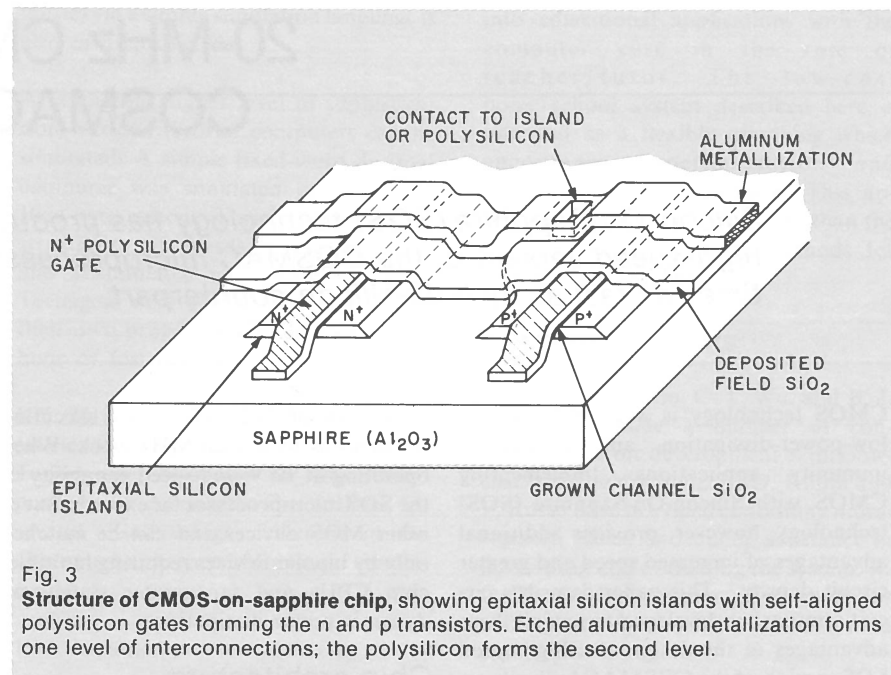


Fig. 3
Structure of CMOS-on-sapphire chip, showing epitaxial silicon islands with self-aligned polysilicon gates forming the n and p transistors. Etched aluminum metallization forms one level of interconnections; the polysilicon forms the second level.

R.G. Stewart has been responsible for CMOS and SOS memory circuit design, including static 1024-bit RAMs and high-density CMOS ROM arrays, since joining the SSTC in 1973. Before coming to RCA, he worked on a number of microwave, bipolar, and CMOS devices.

Stephen Connor has been involved in the computer-aided design of the SOS microprocessor and CCL microprocessor interface circuits since coming to the SSTC in 1974. Prior to that, he worked on the initial process development of the thin-film transistor and silicon-on-sapphire technologies at RCA Laboratories.

Joseph Sinniger came to the SSTC in 1973, working on systems for digital tv tuning and the COSMAC microprocessor. Previous to that, he had been working on the analysis and development of communication systems at the RCA Laboratories.

Contact them at:
Solid State Technology Center
RCA Laboratories
Somerville, N.J.

Left to right: **Stewart, Ext. 6346; Sinniger, Ext. 6253; and Connor, Ext. 7044.**



Second, n⁺ doping of the polysilicon reduced the polysilicon sheet resistance so it could more usefully be employed as a second interconnection level. The improved CMOS-On-Sapphire structure is shown in Fig. 3. The substrate material is single-crystal sapphire of (1102) crystallographic orientation. This material is sliced into 18-mil-thick wafers, which are then ground, polished, and cleaned to remove surface contaminants that could cause transistor source-drain leakage.

Single-crystal silicon with the same crystallographic orientation as the sapphire is grown epitaxially on the sapphire surface to a thickness of 5000 Å. The silicon is then defined and etched into islands, which later will become the n and p transistors. A 900-Å channel oxide is next grown on the silicon islands; this is followed by a 5000-Å-thick polysilicon layer doped n⁺ to minimize its sheet resistance. The polysilicon layer is then etched to form the transistor gates and polysilicon interconnections. The island areas exposed at this step are next doped n⁺ or p⁺ to form the self-aligned-gate, complementary transistors. To complete the structure, a field oxide layer is deposited, contact openings are etched to the silicon and polysilicon, and aluminum is deposited and etched to form the metal interconnection level.

In describing the process, the doping of the silicon under the gate has been omitted because several methods for accomplishing this are currently in use: 1) all-n or all-p doping of the initial epitaxial silicon to produce enhancement transistors of one complementary type and deep-depletion enhancement transistors of the opposite type; and 2) selective n or p doping of the silicon islands to produce normal complementary enhancement transistors of both types. The former method produces transistor thresholds of less than 1 V for both p and n devices, while the latter approach typically produces transistors of higher G_m and sharper threshold characteristics.

The SOS microprocessor was designed to take advantage of the low thresholds offered by the deep depletion process, but can be used with either process variation.

With the SOS process, most of the electrical parameters are similar to those for bulk CMOS fabrication except for the wiring-to-substrate capacitance, which is greatly reduced. With SOS, capacitance to

Table I
Layout design rules (partial listing) for the standard logic cells from which the chip was assembled.

Minimum dimensions for logic cells

Polysilicon gate length—6 μm
Silicon island and transistor width—5 μm
Island-island spacing—8 μm (10 μm oppositely doped)
Polysilicon interconnection width—8 μm
Polysilicon spacing—8 μm
Contact opening—5 × 10 μm (7 × 20 μm for n ⁺ , p ⁺ split contact)
Metal interconnection width—10 μm
Metal spacing—8 μm

the ground plane located under the sapphire substrate is only 0.0004 pF mil for a 5-μm-width conductor; this capacitance is generally negligible. Having nearly eliminated the ground capacitance, one must consider the capacitive coupling between adjacent conductors; this coupling is typically 0.0015 pF/mil for two 5-μm conductors spaced 7.5 μm apart and generally cannot be neglected. This coupling is particularly important when it is into a high-impedance circuit node or between two such nodes.

The microprocessor chip was assembled from a relatively small group of standard logic cells.

Most of the cells were unitized in both the x and the y dimension, corresponding to the metallization interval of 18 μm (see Table I for an abbreviated list of design rules). Unitizing the cells facilitates rapid trial-and-check cell placement procedures that lead to layouts of optimum area efficiency. An example of this approach is the logic layout shown in Fig. 4. The circuitry is assembled from standard cell inverters, latches, and 2- and 4-input logic gates. Each cell of a given type is contained within a rectangle of fixed x and y unit value, the cell blocks are placed on gridded sheets, and the metal and polysilicon interconnects are drawn with the grid spacing. This procedure is repeated using different cell placements until a minimum-area layout is obtained.

Although cell unitization speeds the development of area-efficient chip layouts, each cell must be area-efficient itself to

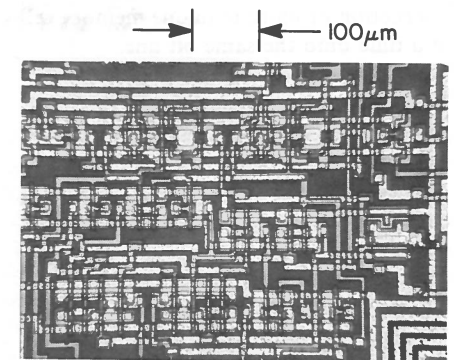


Fig. 4
Section of the control logic, which is constructed of inverter, 2- and 4-input logic gates, and latch cells. The cells are unitized in the x and y dimension consistent with the aluminumization spacing to facilitate minimum-area layouts using trial-and-check cell-placement procedures.

realize a net gain for the approach. For example, 3-input logic gates were found to exhibit poor layout efficiency within the unitization constraints and were therefore not used.

Scratchpad memory design considerations

The 5-transistor storage cell requires less area than a conventional 6-transistor cell, but has an operating disadvantage because the access transistor T1 must operate as a source follower during Read-1 and Write-1 operations. For this reason it was necessary to design the memory carefully and computer-simulate its operation. Fig. 5 shows part of the final computer simulations for Read-0 and Write-1 operations. Protection against memory upset during the read operation is achieved by increasing the Miller-effect capacitance between the output and the internal nodes of the cell. When combined with a high-impedance inverter, W, the internal node is stabilized against upsets caused by transients on the bit line. This effect can be seen in the simulated Read-0 waveforms shown in Fig. 5. To read a 0 state, the capacitance on the bit line is discharged to ground; this must not disturb the logic-0 level stored in the cell. As shown in the figure, the output node in this case is displaced by 2.8 V from its quiescent level, but the interior node is displaced by only 0.9 V, assuring retention of data. Further upset protection is provided by a word-decoder design that limits the voltage risetime at the gate of T1 and also prevents

connection of more than one memory cell at a time onto the same bit line.

During Write-1 operations, T1 source-follower, which limits the speed of the write operation, especially when high n-threshold voltage (T1 is an n transistor) is combined with low V_{DD} . To insure reliable writing over a 3- to 15-V range, p-type transistor T2, as shown in Fig. 2, is placed in series with the write driver so that a Write-1 operation reduces the supply voltage applied to the selected column of memory cells.

This reduces the supply voltage to the cell during writing to approximately one-half its quiescent value within 10 ns. The reduced supply voltage to the cell allows writing over a V_{DD} range of 2 to 17 V, yet does not lead to storage loss in unselected cells if device thresholds are maintained under 1.0 V, which is generally possible with deep-depletion SOS processing.

The complete scratchpad memory, including write-drivers, decoders, and decoder-buffers, occupies a chip area of 66 by 73 mils. At a supply of 10 V, the read-

access time is typically 80 ns, page-mode access time is 8 ns, and minimum write-pulse width is 22 ns.

High-speed interfaces for use with bipolar logic

The high-speed capability of the SOS microprocessor makes it possible to design fast microcomputer systems using a mix of CMOS/SOS and high-speed bipolar logic. Maximum performance in these systems, however, generally requires the CMOS logic to operate at a higher voltage than the 5-V levels typically powering the bipolar logic. A high-speed level shifter is therefore needed to convert bipolar logic levels to CMOS levels. An additional requirement for low-power-dissipation CMOS compatibility is that no dc current paths may exist between the power supplies of the two types of logic or between either power supply and ground.

Fig. 6 shows a circuit diagram of a high-speed level shifter developed to convert T²L logic levels to CMOS levels. Transistors P1, N1, P2, and P3, N3, and P4 form two coupled static latches operating from power-supply levels V_{CC} and V_{DD} , respectively. Transistors N4 and N5 provide dc isolation between the two latches, but permit transmission of digital signals between them.

Test results show operation over a wide range of conditions. If a maximum level shift is required, V_{CC} can be operated as low as 3 V, transforming a logic-1 level as small as 1.0 V at the input to 15 V at the output. In a more typical application with $V_{CC} = 5$ V and $V_{DD} = 10$ V, T²L logic levels of 0.7 V and 2.3 V are converted into CMOS logic levels of 0 V and 10 V, respectively, with a typical delay of 30 ns.

Results

Fig. 7 is a photograph of the SOS microprocessor chip, which measures 5.33 × 5.33 mm (211 × 211 mils), including the 4-mil-wide dicing streets. The scratchpad memory array is located in the upper right corner of the chip. Directly below the memory is the incrementer circuit, then the I, N, X, P, T, and D registers. The internal data bus is distributed through these registers on polysilicon. The external data bus drivers and input level shifters are located at the bottom right, and the control logic and ALU are located on the left side of the chip.

The clocking of the CPU is fully static and uses either an external clock input or an on-chip crystal oscillator. Static clocking permits stopping the clock at any point in the machine cycle. The relationship between instruction time, external-memory access time, chip dynamic power, and clock rate are illustrated by the curves shown in Fig. 8. At $V_{DD} = 10$ V, a maximum clock rate of 20 MHz is possible. At this clock rate, the required external-memory access time (approximately 4 clock cycles minus MA set-up time) is 150 ns. Preliminary results indicate that 30-MHz operation can be obtained at $V_{DD} = 15$ V.

The static design of the chip results in negligible (typically less than 1 mW) quiescent power dissipation. Most of the power dissipation is dynamic, given by $P = fCV^2$, where f is the clock rate, C is the total internal capacitance of the active nodes, and V is the supply voltage (V_{DD}). The capacitance is a function of the logical complexity of the instruction being executed. The power curves of Fig. 8 show averaged dynamic power for "typical" instruction strings, not including V_{CC} supply contributions in the output drivers. The power at $V_{DD} = 5$ V and $f = 8$ MHz is 10 mW, increasing to 70 mW at 10 V and 20 MHz. The V_{CC} supply contributes an additional 30 mW at 20 MHz in driving 100 pF output loads. The total power dissipation and speed/power product are extremely low, even when compared with bulk-silicon CMOS. This is a result of the low capacitance of the internal nodes and also because relatively few of the nodes are active at a given time in the CPU.

Conclusions

Table II summarizes the characteristics of the CMOS-On-Sapphire microprocessor. The 20-MHz clock rate is comparable to that of bipolar logic, such as T²L, but the power dissipation is much less. Also, the complete CPU can be placed on a single chip, in contrast to the several chips required with conventional bipolar logic. The simplified microprocessor architecture and conservative design rules used make the chip suitable for high-volume, low-cost production.

References

1. Meyer, J.E.; Burns, J.R.; and Scott, J.H., Jr.; "High-speed silicon-on-sapphire 50-stage shift register," 1970 IEEE/ISSCC Digest of Technical Papers, p. 200.
2. Young, A.W.; "The CDP1802—a powerful CMOS 8-bit microprocessor," this issue.
3. Russo, P.M.; "Architectural trade-offs in the design of microcomputer systems," this issue.

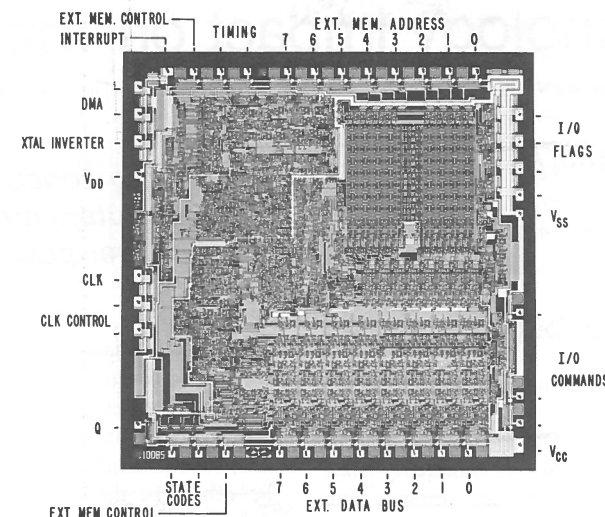


Fig. 7 Microprocessor chip dimensions are 5.33 mm (211 mils) square; chip is mounted in a 40-pin package.

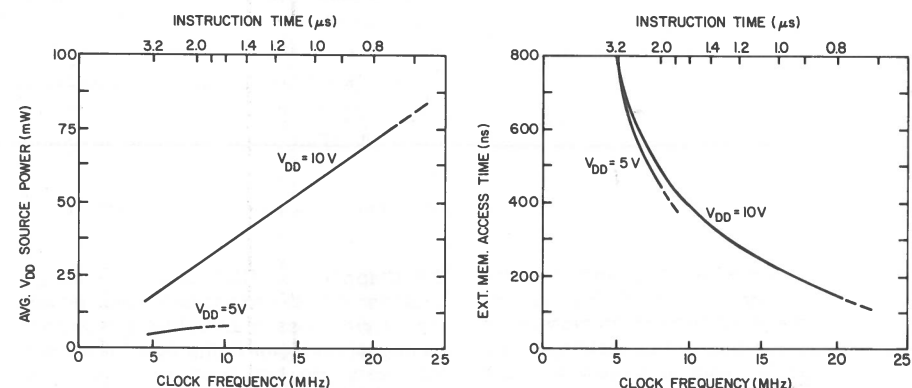


Fig. 8 External-memory access time and power dissipation versus clock frequency; preliminary results indicate operation to 30 MHz is possible at 15V.

Table II Performance characteristics for the 1802S CMOS-on-sapphire microprocessor. Clock rate is comparable to that of bipolar logic but with much less power dissipation.

Chip size—5.33 × 5.33 mm (211 × 211 mils)
No. transistors—4827
Average chip area per transistor—6 × 10 ⁻³ mm ²
Inputs—T ² L-compatible
Outputs—T ² L-compatible
Dynamic performance
Max. clock rate—20 MHz @ $V_{DD} = 10$ V
Avg. V_{DD} source power at max. clock rate—70 mW @ $V_{DD} = 10$ V
—10 mW @ $V_{DD} = 5$ V
Typical quiescent power dissipation—160 μ W @ $V_{DD} = 10$ V
15 μ W @ $V_{DD} = 5$ V
Typical circuit delays at $V_{DD} = 10$ V:
ALU logic—10 ns
Incrementer 8-stage delay—20 ns
Scratchpad memory access time—80 ns
Scratchpad memory write time—22 ns
Average speed-power product—4.0 pJ
Operating temperature range—-55°C to +125°C
(Max. clock rate—24 MHz @ 30°C, 16 MHz @ +125°C)

Fig. 5 Computer simulation of the scratchpad memory, indicating 55-ns read-access time and 37-ns minimum write time.

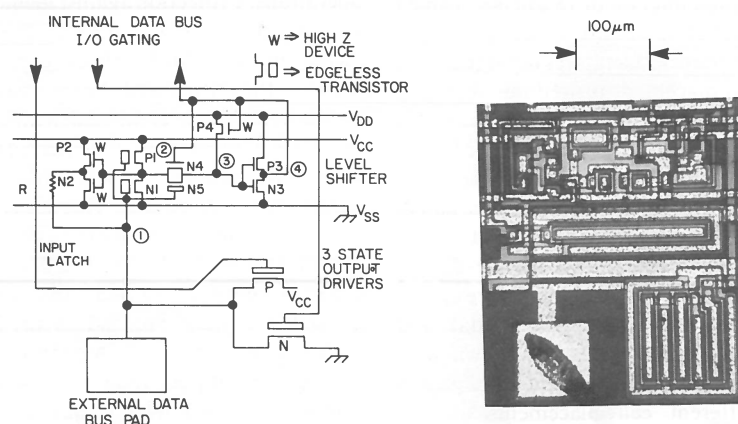


Fig. 6 Input level shifter and associated output drivers for one of the external data bus lines. The level shifter converts TTL logic levels to internal CMOS levels. All gates directly connected to the bonding pad are of an edgeless concentric design to maximize gate breakdown voltage.

Technology impact on microprocessors

W.A. Clapp|A. Feller

Continuing concurrent advances in solid-state technology and computers are pushing microprocessors toward faster speed, lower power, and higher performance.

The microprocessor—developed over the last five to seven years—merges semiconductor advances with computer-design technology. Advances in each of these independent fields stimulate advances in the other. In this environment, the one unchanging factor is constant change. This paper considers future trends of microprocessors, set against a background of changing technology and computer-aided design.

To see the rate of change of technology in clearer perspective, it is helpful to review the time span from scientific invention

Final manuscript received November 24, 1976.

Al Feller came to RCA in 1951 and joined the Computer Division in 1958. He became active in the development of high-speed circuits and interconnection techniques. He has directed the design of more than 50 LSI devices and several LSI computer systems, the most recent constituting the ATMAC CMOS/SOS microprocessor.

Bill Clapp joined RCA as a development engineer in 1965 and has been involved with the interrelationship of LSI technology and its impact on digital computers for the past ten years. He has been responsible for several advanced computer developments including the SUMC series, the B-12, and most recently the ATMAC.

Authors Al Feller (left) and Bill Clapp are examining the 16-bit ATMAC microprocessor developed by their group.

Contact them at: **Applied Computer Group, Advanced Technology Laboratories, Camden, N.J., Ext. PC3276 or PC3257**



until the manufacture of the product for some key past developments. This time lag was about 112 years for photography, 56 years for the telephone, 35 years for the radio, 15 years for radar, 12 years for television, 6 years for the atomic bomb, 5 years for the transistor, 3 years for the integrated circuit, and about 1½ years for the microprocessor. Today we are into the second and third generations of microprocessors.

The result of this change is obvious in the commercial market, where one can see the impact of the personal scientific computer, with its better than 10-to-1 cost reduction in only three years. The digital watch has dropped in price by an order of magnitude, and increased its capability.

Such change is even more sobering in the military market, where it still takes 8 to 10 years from advanced development to fielded equipment. As a result, a manufacturer of military electronics may find himself in the development stage of an equipment when the design approach selected and the parts identified are upstaged by a newly available part which is faster, smaller, and cheaper. The commercial market is faced with product half-lives approaching 1 to 1½ years.

Background of microprocessor technology

In the early 1970s, semiconductor manufacturers continued to increase their capability to place more transistors on a single monolithic chip to the point where they could put a computer on a single chip. Large Scale Integration (LSI) had arrived. This was a simple computer, with perhaps some of it on two or three additional supporting chips, but it was a necessary step to stimulate microprocessor developments.

For years, the semiconductor field had been divided into two camps: one devoted to going faster and faster (the bipolar

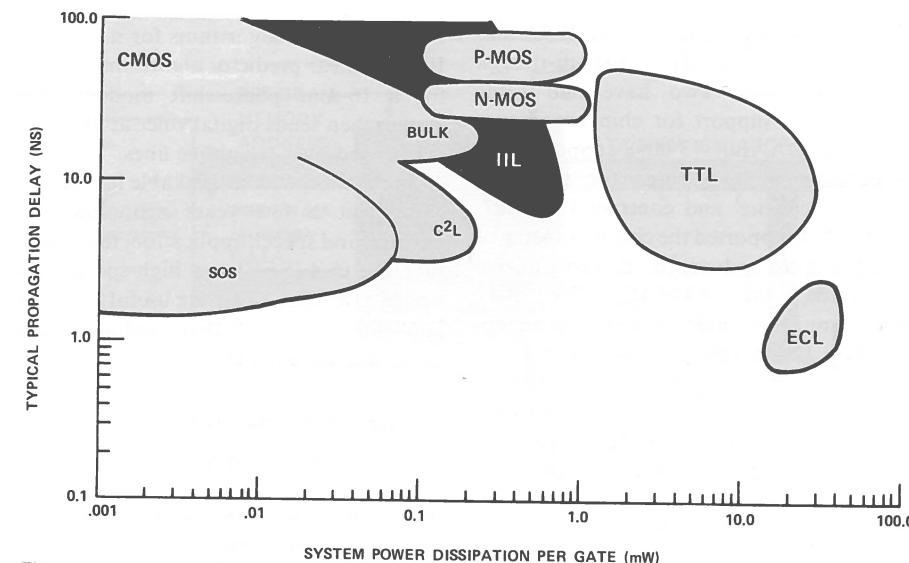


Fig. 1

System speed-power comparison, based on general-purpose logic, such as that in a microprocessor. With this chart, you can calculate the maximum level of integration possible for a given technology running at a given speed.

technology), and the other devoted to putting more and more transistors on a single chip (the MOS technology). Each camp was continually checking on the progress of the other and striving to incorporate the features of the other camp into its own product. As a result, the MOS camp developed technology variants that provided more speed, and the bipolar camp developed technology variants that provided a higher level of integration. These efforts resulted in at least fifty distinct technologies which are present in industry today. The goal of this competition is to provide higher speed and a higher level of integration than those presently available.

A high level of integration must, however, be achieved within a reasonable level of power dissipation; most common packages can handle approximately ½ W. Higher speeds have typically been achieved by higher power dissipation. Thus the ideal technology for LSI and very large scale integration (VLSI) provides the lowest system power dissipation and the fastest speeds. As a result, much attention is now focused on the speed-power product of newer variants of technologies. These speed-power products, however, are usually provided to us based on individual transistor or gate-level data. What is really important is the system-level speed-power product: the speeds achievable at various power levels for a system implemented on a single chip.

technologies in the system context. This figure illustrates the system speed-power comparisons for general-purpose logic, such as that in a microprocessor. It results from considering duty cycles of gates in a general-purpose logic system where all the logic gates don't have to switch at top speed all the time and, in particular, the duty cycle for CMOS logic gates, which dissipate very little power when not switching. From this type of chart, one can calculate the maximum level of integration possible for a given technology running at a given speed.

The increasingly higher levels of integration result not only in the ability to make better microprocessors, but also total system equipment which has smaller sizes, lower power, higher reliability, and which breeds new equipment concepts not conceivable before LSI. In addition to these advantages, the cost per gate has been steadily declining because fabrication of LSI and VLSI is basically a batch process. Therefore, the more transistors made per step, the cheaper the resulting transistors become.

The benefits cited above are from the point of view of the system builder. On the other hand, semiconductor vendors want high-volume production on as few parts as possible. In fact, engineering is figured as 5 to 10% of sales. So, for each dollar of engineering time in designing a custom LSI circuit, a company needs \$10 to \$20 of parts business from chip production. The solution to this situation was a universal

programmable LSI chip in the form of a microprocessor, adaptable to a wide market. As a result, there are now 30 to 35 unique microprocessors, plus their supporting chips. The microprocessor also created a large demand for another universal chip—the random access memory—which was already on the shelf.

Impact of computer-aided design

Computer-aided design (CAD) will also play a large role in future trends of microprocessors. CAD uses a set of computer programs to automatically lay out and generate the artwork to fabricate a custom-design LSI array. The Army (through ECOM at Fort Monmouth*) and RCA during the last several years have advanced CAD for LSI and VLSI complexities. The most successful approach has been the standard-cell CAD concept. Standard cells are basic building blocks which are of standard height and variable width to accommodate a wide variety of cell functions. These cells are designed, checked, and placed on magnetic tape for future reference. The logic designer then partitions his logic design into these standard cells and gives an interconnection list representing the desired logic function to the computer, which proceeds to automatically place and completely interconnect the cells selected to implement the desired logic. Magnetic tapes from CAD programs drive artwork generation equipment to make masks which are then used to fabricate LSI arrays.

In this way, CAD enables the system designer to use LSI without bearing its high non-recurring costs. CAD can reduce the non-recurring chip-development costs by a factor of 3 to 5 times, depending on the logic to be implemented. Custom-designed chips can be used independently of, or in conjunction with, existing off-the-shelf parts, including microprocessors, to acquire the system advantages provided by advances in technology. These advantages include reduced size, increased reliability, and higher performance, all at a lower cost.

Recently, a CMOS/SOS chip—an 8-bit slice of the RCA-developed ATMAC processor—was designed with CAD techniques and fabricated under contract DAAB07-75-C-1314. The 4560 transistors on this single chip yield about 1500 logic

*Several contracts have been involved, notably DAAB07-74-C0176.

gates. Many general-purpose computers built in the 1960s used 5000 to 10,000 logic gates and required two racks of hardware. Today these racks can be replaced by four to six LSI chips requiring only 0.001 of the power.

Current microprocessors and microcomputers

There has been much confusion in distinguishing between microprocessors and microcomputers; Fig. 2 clarifies the differences. In practice, the microprocessor may be less than 10% of a total microcomputer system. In both commercial and military applications, the microcomputer system, rather than the microprocessor, is of primary importance, but the microprocessor has been receiving all the attention. The same technology and techniques will continue to be applied to memories, control and timing logic, and peripheral interfaces to reduce the total cost of a microcomputer system.

Custom-design microprocessors may be tailored to an application or class of applications. The advantages of custom-design microprocessors over commercially available microprocessors are 1) an order-of-magnitude improvement in performance and 2) availability of a processor one to two years sooner. In fact, the equivalent may never be available commercially. Only with CAD is this concept feasible for the small-volume applications typical of the military market. The concept is also cost effective for large-volume applications during initial phases where the customer is likely to frequently change his requirements. Then as volume warrants it, a custom-design equivalent can be prepared.

During the past four years, RCA has completed several custom-design microprocessors. Two have had some government support for chip processing. Contract DAAB07-73-0098 supported the processing of two chips for the B-12 microprocessor,¹ and contract DAAB07-75-C-1314 supported the chip processing of RCA's most advanced custom microprocessor, the ATMAC. The B-12 processor, fabricated in 1973, is an all-CMOS, 12-bit-wide processor with 47 instructions (including multiply and divide); it has built-in clock generation and bus-driver circuits. Available about two years ahead of comparable commercial microprocessors, the B-12 was incorporated into an Army Radiac demonstration unit (as a microcomputer) which provided both total-radiation-dose and dose-rate readouts with pre-settable alarm levels for each.

A more recent design is the ATMAC, a CMOS/SOS processor optimized for array-type calculations. In a typical signal-processing application, ATMAC provides a throughput of about 8 million instructions per second. This performance is an order of magnitude more than is available in the new AN/UYK-30, a standard military minicomputer. ATMAC is also bit-slice modular in 8-bit slices. A typical 16-bit version is on four chips, provides 189 instructions, and requires only 3/4 W. Fig. 3 shows ATMAC as a component of a generalized microcomputer system. The program memory is ROM storage, the data memory is RAM, and the I/O devices are selected for a particular application. One present application has ATMAC at the heart of a narrowband speech system. Here the AT-

MAC executes algorithms for not only a 10-pole linear predictor algorithm but also for a 16-tone phase-shift modem. The system then sends digital voice at 2400 bits per second over telephone lines. This type of application was unthinkable for tactical equipment a few years ago. For this narrowband speech application, the special function unit (SFU) is a high-speed multiplier, but other SFUs are useful for other applications to further enhance the capability of ATMAC.

Future trends in microcomputers

Handheld processors today outperform many racks of equipment made just a couple of years ago. The future trends of microcomputers are in technology development and computer architecture. It is reasonable to project continuing advancement in the level of integration possible on a monolithic chip; a growth of five to ten times improvement in the next five to seven years is not unreasonable. This means from 50,000 to 100,000 transistors on a single chip, and underlines the necessity for lower-power technologies to keep the chip dissipation around 1/2 W—an average of about 10 μ W/transistor in the system.

At the microprocessor level, three distinct trends are developing: The first is the move from 8-bit-wide processors to 16-bit-wide processors. A second trend is toward the integration of the processing unit, some read/write memory and read-only memory, and some peripheral control all on the same chip. A third trend is that of developing custom-design microprocessors which are used either alone or in conjunction with other off-the-shelf microprocessors to implement higher-performance or special-purpose systems. Thus, these new technologies and different organizations will allow use of additional logic to make processors faster and more powerful.

For military applications, the search is on for standards in this evolving microprocessor world. Spending more than \$100 million annually on semiconductors, the military and prime system contractors are incorporating commercial microcomputers in future equipment. From this effort, *de facto* standards are now surfacing; three of the more often mentioned are the Intel 8080, the RCA 1802, and the Motorola 6800. Most applications to date, however, are either

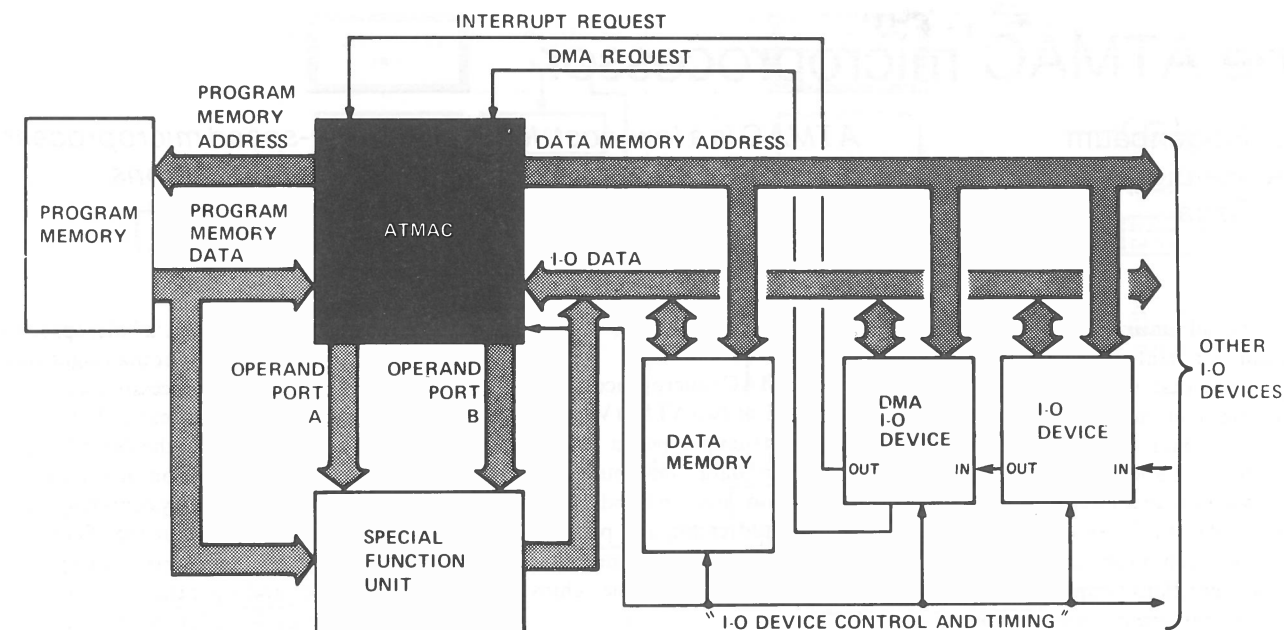


Fig. 3
ATMAC microprocessor as a part of general microcomputer system. At present, ATMAC is RCA's most advanced custom microprocessor.

controllers or dedicated real-time data processors. The military is also emphasizing requirements for second- and third-sourcing, and for full military-specification parts for not only the microprocessors but also memories, I/O devices, and other components.

At the system level, there is a trend away from a complex, centralized, data processor and toward microprocessors as controllers for special-purpose black boxes which themselves might use custom-LSI or custom-design microprocessors. This concept leads to distributed processing where the total job is accomplished by a number of small processors, each dedicated to part of the total problem and each working on its own segment of the problem.² Typical studies in this field have already been undertaken by the Air Force in its Distributed Processor Memory efforts and by NASA-JPL in its Unified Data System. Older concepts, such as ILLIAC IV, are dwarfed in comparison to what is now possible.

A total system could include anywhere from one to several hundred microcomputers, adding considerably to the reliability and flexibility of the system. In the limit it leads to putting software into hardware where each microcomputer is executing a dedicated segment of the total application software. This trend will also aid the expensive and growing problems of software development costs.

In regard to software, assembly code is most frequently used for dedicated functions. As the inherent speed of microprocessors increases, and the cost of memory decreases, the trend will be toward the use of more standardized higher-level languages such as Fortran, CMS, PL/M, and DOD-1. Compilers for these higher-order languages are under development. However, PL/M compilers already exist for the Intel 8080 and the Motorola 6800.

Summary

The major trends discussed here can be summarized as follows:

- Microprocessors and microcomputers will become the building blocks of future systems.
- The microprocessor market will condense to 8 to 10 types of microprocessors from the 30 to 40 types presently available.
- A move to a 16-bit data path microprocessor is a natural evolution.
- There will be a move to minimize support chips by including such things as clock generation logic, and both read-write and read-only memories on the same monolithic chip.
- Customized microprocessors are available today which can execute 8 million instructions per second, and this performance will increase by a factor of 2 to 4 in the next 5 years.

- Use of multiple microcomputers will increase in two directions—1) dedicated hardware performing dedicated functions, and 2) more general-purpose distributed and array configurations where several hundred microcomputers will be interconnected for a given application.
- Future systems will use off-the-shelf LSI microprocessors and support parts where possible, perhaps supported by custom-design LSI.
- However, for higher performance, future systems will require more custom-design LSI (including custom microprocessors) supported by off-the-shelf microprocessors and other parts where their slower speed can be utilized.

References

1. Ozga, S.E.; "The B-12—an automotive microprocessor," RCA reprint RE-22-2-6
2. Russo, P.; "Architectural trade-offs in designing microcomputer systems," *this issue*.
3. Clapp, W.A.; "LSI computer design—SUMC/DV," RCA reprint PE-569.
4. Feller, A.; "LSI computer fabrication—SUMC/DV," RCA reprint PE-569.
5. Feller, A.; "Design automation techniques for custom LSI arrays," AGARD Lecture Series No. 75 (Apr 1975) pp. 7.1-7.15.
6. Merriam, A.; and Zieper, H.S.; "Simulation: methodology for LSI computer design," RCA reprint PE-569.
7. Feller, A.; "Automatic layout of low-cost quick-turnaround random-logic custom LSI devices," 13th Design Automation Conference, San Francisco, CA (Jun 1976) *Proceedings* pp. 79-85.
8. Feller, A.; and Noto, R.; "Random logic custom LSI and VLSI using the standard cell approach," 1976 GOMAC, Orlando, FL (Nov 1976).

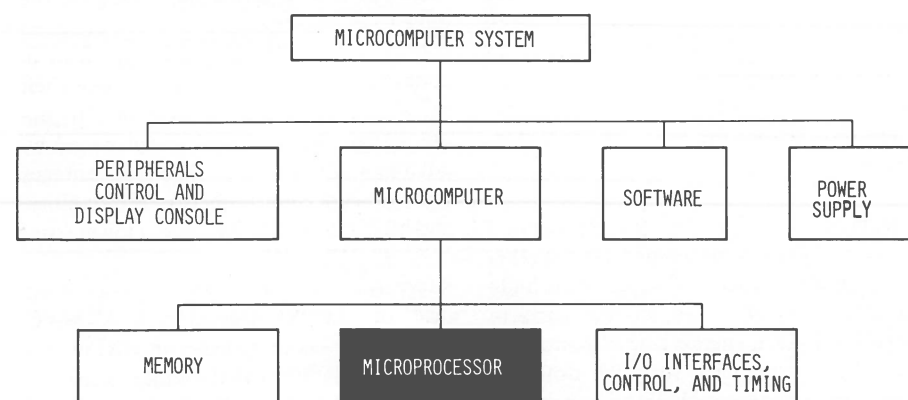


Fig. 2
Typical microcomputer hierarchy. This drawing places the microprocessor in system context with memory, input/output devices, software, and support circuits.

The ATMAC microprocessor

A.D. Feigenbaum
W.A. Helbig
S.E. Ozga

ATMAC is a low-cost, low-power high-speed microprocessor designed specifically for array-type computations.

Conventional microprocessors are impractical for real-time signal-processing applications that require very high-speed processing and accessing of very large arrays of data. Recently, Advanced Technology Laboratories (ATL) has developed a bit-slice microprocessor called ATMAC specifically as a low-power, low-cost, high-speed means of solving matrix-algebra algorithms, implementing digital filters, and performing other computations requiring high-speed signal processing.

What is ATMAC?

The ATMAC microprocessor (Fig. 1) is composed of two VLSI (Very Large-Scale Integrated) chip types: a "data execution unit" for data manipulation and an "instruction and operand fetch unit" for memory addressing and program control. These chips are partitioned into eight-bit slices; however multiple chips can be

Final manuscript received January 24, 1977.

Alan Feigenbaum recently joined the Engineering Communications section of RCA's Advanced Technology Laboratories, where he is involved with writing reports and proposals for various projects undertaken by the ATL group.

Contact him at: **Engineering Communications, Advanced Technology Laboratories, Camden, N.J., Ext. PC 2853**

Walt Helbig has been working in computer technology since he joined RCA in 1952. For the past two years, he has been Engineering Leader of the Architecture and Applications Group directing work in the areas of system design for LSI microprocessors, multiprocessors, fault tolerant processors, array processors, and associative processors. He is currently directing hardware and software design and implementation of the ATMAC microprocessor and CCD mass memory.

Contact him at: **Architecture and Applications, Applied Computer Systems Laboratory, Advanced Technology Laboratories, Camden, N.J., Ext. PC 5071**

Stan Ozga has been a major contributor in the design of several LSI computer systems since he joined RCA's Advanced Technology Laboratories in 1970. He is currently project engineer of the ATMAC microprocessor, with prime responsibilities in architecture design, detailed logic implementation and system integration.

Contact him at: **Architecture and Applications, Applied Computer Systems Laboratory, Advanced Technology Laboratories, Camden, N.J., Ext. PC 3094**

Authors (left to right) **Feigenbaum, Helbig, and Ozga**. The control unit on the table is used for ATMAC program development.



assembled to construct a microprocessor with word lengths greater than eight-bits. A typical 16-bit microprocessor (see Table I) would require two of each chip type. In other cases, because of the flexibility of the design and the inclusion of several "byte transfer" instructions, systems may be configured where more than one of either chip type may be used. For example, two instruction and operand fetch unit chips may be used for addressing up to 65,536 words of data memory and program memory, and three data execution unit chips may be used to permit handling single-word data with precisions of up to 24 bits.

ATMAC is also capable of a full computational repertoire, with throughput of millions of operations per second. The total amount of hardware necessary for a system is minimized, because ATMAC uses an eight-bit-slice partition for the two VLSI chips and bidirectional buses for interconnections with peripheral equipments and subassemblies. The bidirectional bus system (Fig. 2) reduces the number of connections between ATMAC and other units and is cost-effective. It allows ATMAC to communicate asynchronously with program memory, data memory, I/O devices, and special function units (such as a hardware multiplier or accumulator).

Additionally, ATMAC's architecture permits many operations to be performed in parallel. This provides ATMAC with substantial gain in processing throughput over other microprocessors that have equal or faster cycle times. It is organized to provide maximum flexibility in configuring a system for various applications, and can integrate into its operation one or more unique SFUs designed for a particular application.

ATMAC also has the ability to perform fast-fourier transforms (FFTs), which allow information to be extracted from signals in the time domain by transforming them into the frequency domain. The modular sections of an FFT program have been written for the ATMAC

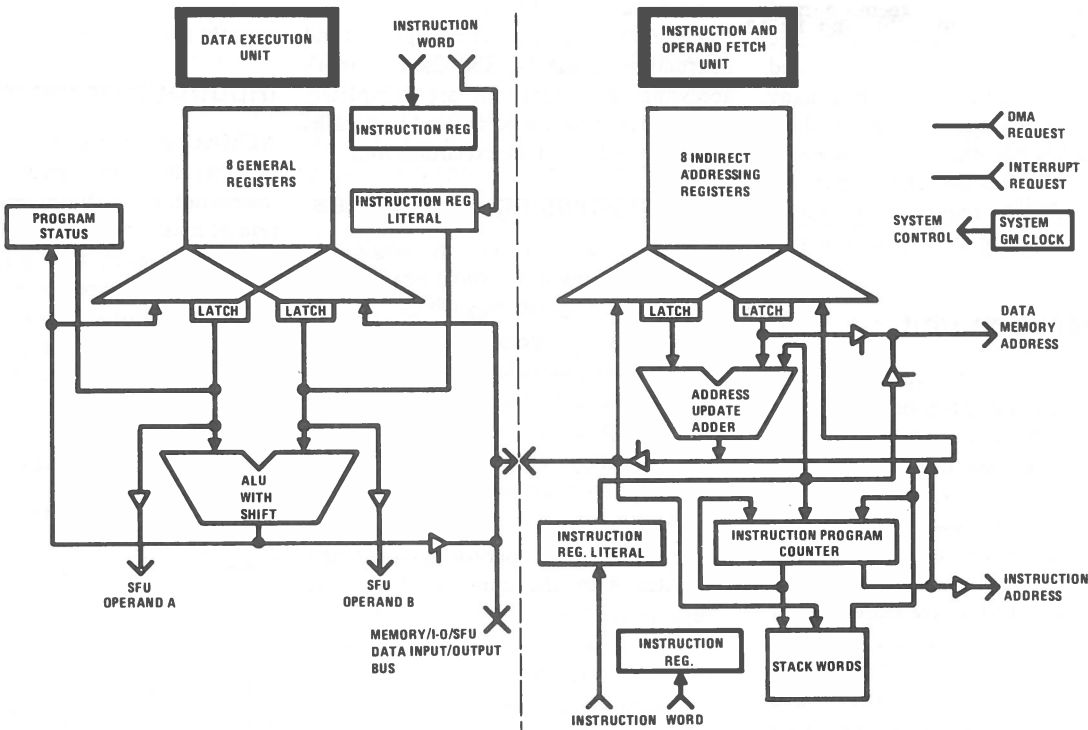


Fig. 1 ATMAC microprocessor consists of two chips: the data execution unit and the instruction and operand fetch unit.

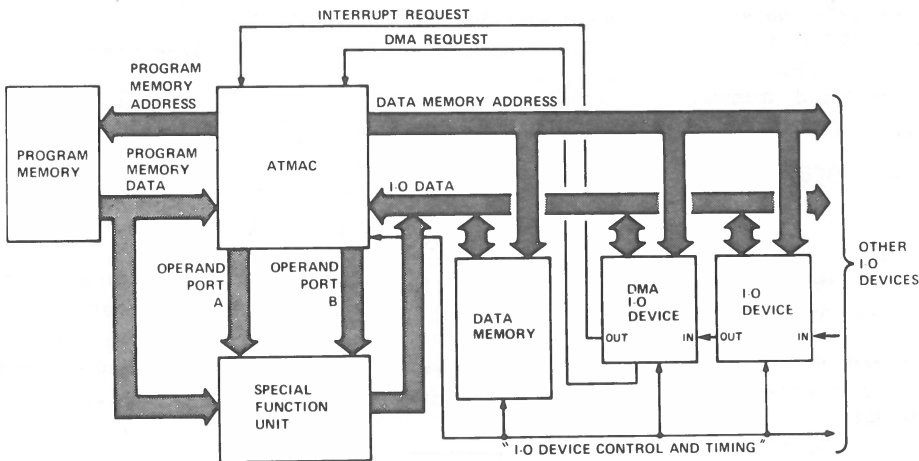


Fig. 2 ATMAC bidirectional bus system allows the microprocessor to communicate asynchronously with memory, I/O, and special function units.

Table I
Typical 16-bit ATMAC system consisting of two data execution units and two instruction and operand fetch units.

Data word length	16 bits (8-bit slices)
Memory address length	16 bits (8-bit slices)
Data memory	65,536 16-bit words
Program memory	65,536 16-bit words
Register complement	eight 16-bit general registers eight 16-bit indirect address registers one 16-bit program sequence counter four 16-bit program counter stack registers
Number of instructions	189
Instruction execution time	short cycle—280 ns Intermediate cycle—350 ns
DMA channel	One word every 700 ns
Support software	Fortran IV cross assembler Fortran IV simulator Scientific subroutines
Power dissipation	≤½ W at 3×10 ⁶ instructions/s

Table II
Fast-Fourier transform memory requirements and execution times for an ATMAC microprocessor with SFU multiplier.

No. of complex points	Data memory words	Program memory words	Execution times (ms)
4	32	16	0.0167
8	64	40	0.0519
16	128	88	0.1427
32	256	184	0.3638
64	512	376	0.8819
128	1024	760	2.0724
256	2048	1528	4.7598
512	4096	3064	10.7435
1024	8192	6136	23.9315
2048	16384	12280	52.7457

microprocessor so that the execution time, I/O time, scaling time, and memory-size requirements for both the program and data memory can be estimated. For many applications, the execution times are short enough to allow real-time FFT processing (see Table II). Results show that the ATMAC memory requirements for a moderate number of data points is very reasonable.

Data execution unit

The data execution unit performs arithmetic and logic operations defined by ATMAC's instruction repertoire. All the operations performed in this unit have operands derived from either the general register stack, an immediate operand from the instruction word, or a hard-wired literal. The operations are register-to-register oriented; results are stored in one of the selected registers and optionally sent over the I/O data lines to I/O devices, to data memory, or to the instruction and operand fetch unit (see Fig. 1). The main architectural features of the data execution unit include the arithmetic and logic unit, which performs all arithmetic operations specified by the executed instructions, and the general register stack, which is a quad-port, eight-word, scratch-pad memory used for arithmetic operand storage.

Instruction and operand fetch unit

The instruction and operand fetch unit is the control portion of the ATMAC microprocessor. It includes the following:

Indirect address register—a quad-port, eight-word scratch-pad memory used for indirect address storage;

Address update adder—a two's complement arithmetic unit used optionally to update the indirect addresses that are transmitted onto the data memory address lines;

Instruction program counter—a full length, double-rank counter responsible for instruction sequencing control, may be set with branch addresses;

Instruction program counter stack—a four-word last-in-first-out, register stack used for interrupt, subroutine, and iterative loop program linkages,

System clock;

DMA synchronization logic;

Interrupt control.

As in the data execution unit, the instruction and operand fetch unit is byte-

expandable. Among its functions, it includes instruction sequencing, operand and instruction accessing, and data memory accessing. Its operations are completely independent and are performed in parallel with those of the data execution unit.

Data processing capabilities

ATMAC can perform, in single-step operations, many of the functions possible on one or two variables, such as:

The arithmetic operations, add and subtract, with options for handling double-precision arithmetic and multiply-and-divide step functions;

The logical operations *and*, *or*, *exclusive or*, and *complement* (either ones or twos),

The register-load operations—loading a register with the content of another register (copy), zero, positive one, and one more than, or less than, the content of another register,

Absolute-value operations—placing the absolute value of the content of one register into another, and the addition of the absolute value of the content of one register with the previous content of the register being used, to accumulate the sum.

ATMAC can also perform, with the use of one or two instructions, all of the possible test conditions on zero, one, or two variables and branch to a new point in the program execution if the test is successful. Further, with the use of one type of SFU, ATMAC can multiply two integers at high speed, multiply two fractions, accumulate the sum of successive double-precision products of the multiplication of two in-

tegers or fractions, and transfer the results directly into data memory.

Input/output capabilities

ATMAC is capable of transferring data to peripherals by the direct execution of an instruction or as an optional parallel function of many instructions. Direct transfers of data between two peripherals is accomplished by direct execution of an instruction, optional parallel I/O function of many instructions, execution of the direct memory access cycle, or an interrupt-driven I/O.

Memory addressing

Data memory (I/O device) addressing is done by using the content of one of eight indirect address registers, or by using a direct address contained in the immediate operand field of some of the ATMAC type II instructions. When the content of an indirect address register is used as the address, it may (through the use of a parallel operation) be modified so that it will have the proper value needed for its subsequent use.

Program execution control

ATMAC uses three types of hardware elements to control the execution of programs. These are the instruction program counter, the instruction program counter stack, and the iteration counters.

The instruction program counter is a double-rank counter that contains two addresses. The first rank contains the address of the instruction presently being executed and the second rank contains the

address of the next sequential location in the program memory. A four-word, last-in-first-out register stack controls the program in nested loops, subroutines, and interrupts. When the present program sequence is broken by a transfer to some other routine, the content of the second rank of the instruction program counter may be "pushed" into the stack, thus moving all of the previous contents of the stack's four words down one location (discarding the previous content of the fourth word). For machine configurations using address lengths of sixteen or more bits, two iteration counters are provided. Separate controls are available for each of these counters so that, under control of the options provided in the instruction format, these counters may be individually tested and decremented.

Software support

In addition to the ATMAC microprocessor hardware, RCA has developed a complete software support package. An off-line cross assembler has been developed and is operational on the Univac Series 70/45 and DEC PDP 11/40 computers. The assembler is written in Fortran IV and is capable of translating user symbolic source code to object code, suitable for loading into the ATMAC program memory. The object code is loaded on 9-track magnetic tape and is also used as an input to the off-line instruction-level simulator, which is also written in Fortran IV. It is capable of testing the assembled software independently of the microprocessor and can determine how long it takes to run a simulated program. The simulator can view the computer state after each instruction or group of instructions and determine the content of each memory and register location. The major features of the simulator are user command language, detailed instruction simulation, I/O device simulation, DMA device simulation, output reports, debugging, and SFU simulation. In addition, a Fortran IV computer language and various mathematic routines are being written.

Applications

One of the first ATMAC applications was in the RCA narrowband digital speech processing system shown in Fig. 3.

This system required a high-performance processor that could perform all the arithmetic and logic functions of a secure terminal operating in a full-duplex mode, including the processing of the linear

predictive coding (LPC-10) algorithm. RCA examined the major microprocessors on the market and concluded that none had the architecture or speed to perform the LPC-10 algorithm in real time. In fact, these microprocessors, including the bipolar bit-slice microprocessors, proved to be 2½ to 3 times slower and more expensive than ATMAC in speech-processing applications. ATMAC had the architecture that provided the type of parallel operation and array processing necessary for performing the LPC-10 algorithm in real time.

ATMAC is well adapted to controlling the multiple use of a network of communication channels.

This application enables time sharing (time division multiple access—TDMA) of transmission among a number of users. Control and use of a channel depend primarily on the operational needs of the particular group of users who have access to it. An entire subscriber unit can be controlled by ATMAC, which, by means of an address and data bus, controls other programmable hardware used for signal processing. In the receive mode, ATMAC performs a digital "automatic gain control," which maintains a selected output level. In the transmit mode, ATMAC can adjust the precise carrier frequency to ensure that the transmitted signal is received by satellite equipment with the same frequency as the channel control order wire (CCOW) transmission. CCOW allows operator access to transmission channels and other users to join the transmission net.

An avionics computer for a space-guidance control system is another unique application of the ATMAC microprocessor.

One of the requirements for such a system is that the microprocessor should be able to handle long word lengths—24 bits or more. Most commercial microprocessors (which can only handle 8- to 16-bit word lengths) are not expandable and, therefore, could not meet this particular requirement. ATMAC, on the other hand, can have at least a 24-bit word length by using three 8-bit-slice chips.

Another requirement for this type of system is that the microprocessor should be compatible with a flight-qualified core memory and simultaneously be able to execute elementary instructions. Since most flight-qualified core memories have a cycle time of 800 ns or greater, an efficient architecture with a fair degree of parallelism is required. ATMAC's efficient

and high-speed throughput is able to meet this requirement.

ATMAC can provide a guidance control system with notable advantages in accuracy, reliability, vehicle attitude determination, redundancy mechanization, computer throughput and flexibility, and system growth potential. It can also serve as a preprocessor for a guidance-control system and perform thorough on-line self-check diagnostics.

RCA has proposed to use the ATMAC microprocessor in a fingerprint matching system for the FBI.

The FBI's current Automatic Fingerprint Matching Processor, which uses the M-40 fingerprint algorithm, has an average matching time, per pair of prints, of about 20 ms. The FBI would like to have a unit that could use the M-40 algorithm to perform four pairs of comparisons per hand in less than 10 ms. For this application, RCA has taken advantage of ATMAC's multiprocessor capability. While a single ATMAC microprocessor could perform the program execution in about 25 ms, a four-microprocessor system is capable, on the average, of processing the entire fingerprint-matching operation for one search-and-file comparison in 10 ms.

Several other applications have been proposed.

ATMAC can be employed in a Global Position Satellite system, where it would control range tracking loops, select a satellite for a navigation position fix, perform a frequency and time search for a time transmission (FFT), and give range readings for navigation position fixes. In other applications, ATMAC can be the basis for high-speed, low-cost, efficient, and flexible signal-processing systems for such projects as sonar signal processing, image bandwidth compression, and seismic data processing.

Conclusion

Because of its advanced architecture, expandable word length, ability to perform complex array computations, and interface with SFUs and other peripheral equipment, ATMAC can be adapted to almost any type of signal processing system at a very low cost. Incorporating the latest advances in CMOS/SOS VLSI technology and computer-aided design, this exceptional microprocessor may prove to be a trend setter in the field of signal and data processing.

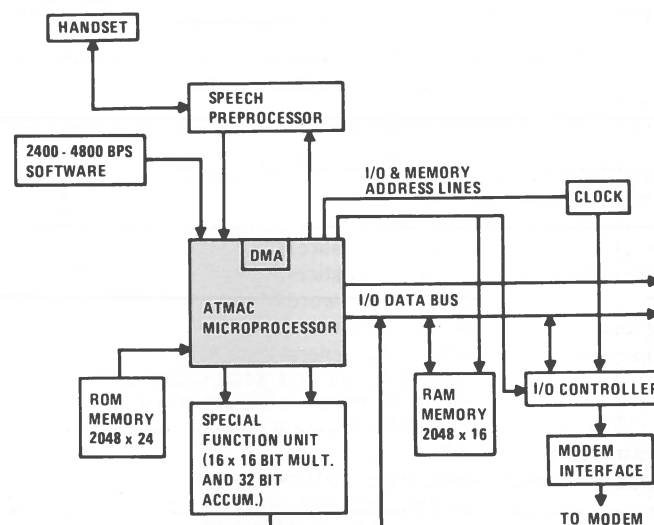


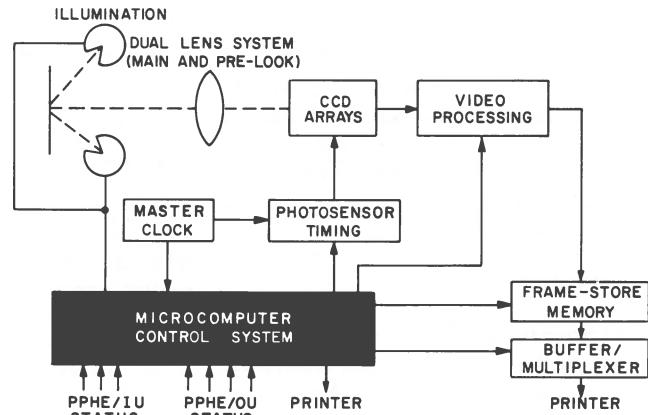
Fig. 3
Narrowband speech processing system uses ATMAC to provide the type of parallel operations and array processing needed.

How RCA engineers are using microprocessors

A quick survey throughout RCA shows that microprocessors are being put to work in highly divergent end uses.

Proposed message system to read 10 pages/second

Recently, RCA proposed a program to develop a scanner subsystem that would convert incoming messages to electrical signals for an electronic message service system. This system is designed to perform a vital function for printing and paper-handling equipment (PPHE)—reliably read incoming messages at rates up to ten pages per second. RCA has proposed to use the COSMAC CDP1802 microprocessor controller to monitor the interface signals to and from the PPHE input unit (IU) and PPHE output unit (OU), and then control the scanner subsystem processing, storage, and outputting of data. The microprocessor would provide the necessary command and status signals to maintain a smooth flow of valid documentation information. Decision-making programs would be stored in ROM, so modifications in scanner-subsystem operation could be made by simply reprogramming the ROM.



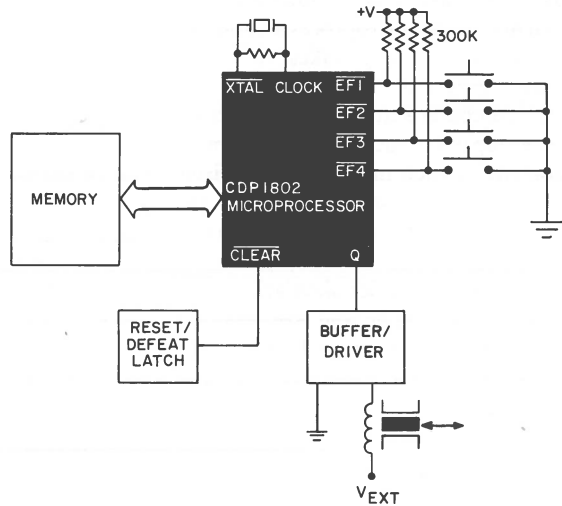
Alan Feigenbaum
Advanced Technology Laboratories
Camden, N.J.
Ext. PC-2853

Digital combination lock has 10⁹ reprogrammable combinations

Requiring only five integrated circuit packages, this electronic lock has its combination entered via four pushbutton switches. The debouncing of the switches is handled by the software and only the 4 "tumblerless" switches are visible to the user. The input code is compared against the combination bank by the software, and an error forces the entire sequence to be reentered to open the lock. Internal timing chains force a reset if an excessive time delay occurs between successive inputs. Total current drain is minimal (5mA typical at V_{DD} = 5V and a 1.6-MHz clock), providing battery holdup for extended periods in the event of a power failure.

The heart of the lock is the RCA CDP1802 COSMAC 8-bit microprocessor. The combination code is entered via the 4 serial-input EF lines. The lock mechanism is driven by a high-current buffer/driver, which is controlled by the Q flip-flop output of the CDP1802. The Q bit is set or reset under program control in response to the input code.

The number of possible combinations (p) is a function of the available memory storage locations (x), where $p = 4^x$.



This lock has 4¹⁵ or 1.07 X 10⁹ possible combinations. An application note describing the digital combination lock is in progress.

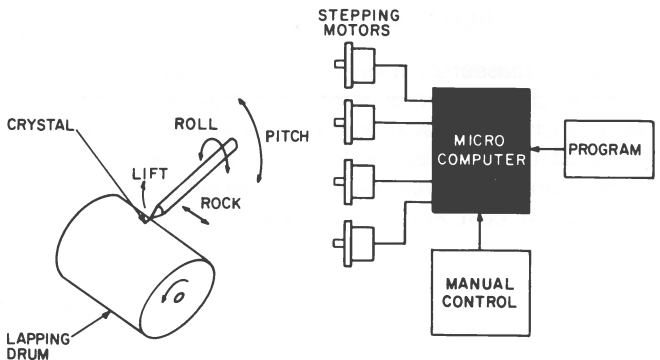
William Clark
Solid State Division
Somerville, N.J.
Ext. 7364

Manufacturing

Programmable microcomputer shapes VideoDisc crystal

The stylus for playing the RCA VideoDisc is a minute sapphire crystal that must be machined with extreme accuracy. The machine that accomplishes this task controls the crystal's four main motions, whose timing and sequencing must be highly precise and repeatable.

A newly designed production lapper uses four stepping motors whose motions are dictated by a programmable control system (Texas Instruments' 5T1). The microcomputer provides the precision and repeatability required. In addition, making changes in the program permits modifications in the crystal shape and the generation of new shapes.

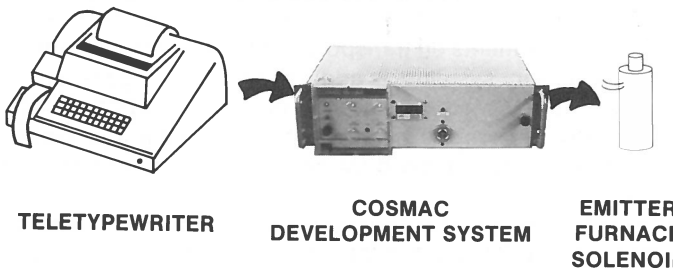


B.E. Mesa
SelectaVision Project
Indianapolis, Ind.
Ext. VR-3237

Microcomputer controls bipolar IC emitter furnace

A COSMAC Development System has been controlling a diffusion furnace at the Solid State Division's Findlay, Ohio plant since June 1976, three shifts a day. Stored in the system's memory are over 100 diffusion schedules for the different device types presently in production. When the operator inputs the device types making up the furnace load via the teletypewriter, the system checks the schedules to make sure that the entire furnace load requires the same schedule. Error message and alarms prevent incorrect combinations.

If all the device types in the load call for the same diffusion schedule, the microcomputer types out a RUN PROCESS message and indicates what the schedule will be. The load is then put in the furnace tube, a button is pushed, and the furnace time-out is under control of the microcomputer. A solenoid is actuated at the proper time to introduce dopant



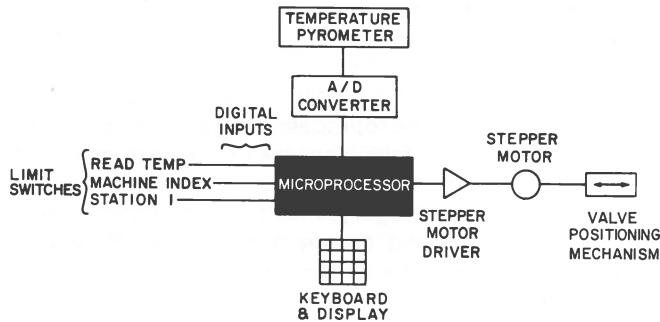
gases into the furnace tube. An LED display on the front panel indicates time remaining; when the process is complete, an audible alarm alerts the operator.

Luke Dillon
Manufacturing Systems and Technology
Cherry Hill, N.J.
Ext. PY-4381

Microprocessor control for glass forming

An indexing glass-forming machine used at the Picture Tube Division's Circleville plant has eleven stations, with a mold at each station. Although cooling air is applied to each station, the air-control valve is adjustable only at one station and so remains at that position for the other ten. Also, the mold temperature can only be read at one station.

To control the temperature of each mold, the temperature history of the individual molds must be stored, along with their unique control-valve positions. Then, when a mold indexes to the control station, the microprocessor performs calculations to determine the adjustment for that cooling valve.



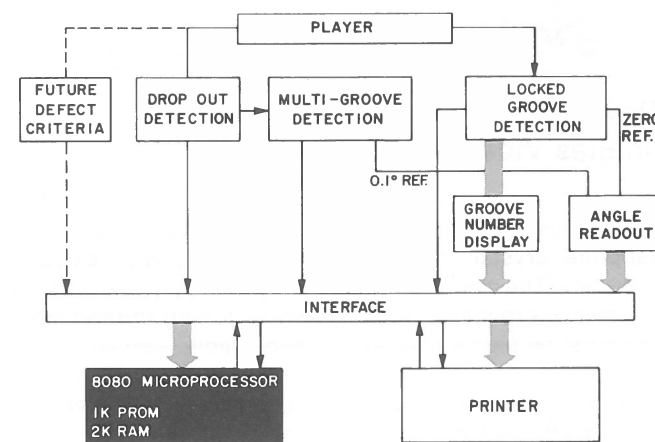
Richard W. Marshall
Picture Tube Division
Circleville, Ohio
Ext. 312

Testing

Microprocessor control for VideoDisc testing

A microprocessor-controlled test system is being designed to locate the position of defects that may be present on the surface of a VideoDisc. When a specific type of defect is detected, the radius and angle of its location are printed out. A microprocessor was chosen for this application because of its speed, cost, and data-storage and data-manipulation capabilities.

A three board Pro-Log microprocessor system with an Intel 8080 CPU was chosen because complete card systems that were compatible with existing test systems were available with PROM capability.



J.W. Stephens
SelectaVision Project
Indianapolis, Ind.
Ext. VR-3088

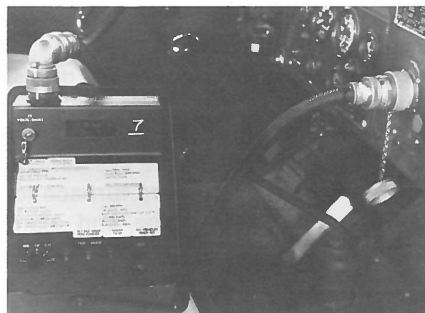
Microprocessor system tests vehicle engines rapidly and accurately

The STE/ICE (simplified test equipment/internal combustion engine) system performs more than 45 types of tests and measurements on transport and tactical vehicle engine and accessory systems. In addition to measuring pressure, temperature, speed, voltage, and current, the system electronically performs a power test on gasoline and diesel engines without the need for an external dynamometer.

Using the test meter shown here with a Diagnostic Connector (connected to built-in transducers and test points in the vehicle), a mechanic needs only a few minutes to perform a series of tests that usually takes hours using conventional test equipment. This flexibility in a hand-held vehicle test meter is achieved by applying a microprocessor for

measurements, conversions, displays, and solving test algorithms. The first prototype units, which were designed in 1973, used the Rockwell PPS-4 microprocessor. The production design, which will be fielded in 1978, uses RCA's COSMAC CDP1802 microprocessor.

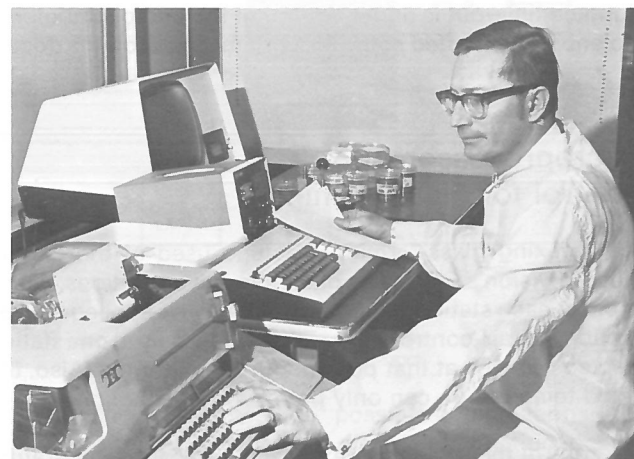
R.T. Cowley
Automated Systems
Burlington, Mass.
Ext. 3185



Programmable calculator produces savings during transistor manufacturing

In the manufacture of homotaxial transistors, it is necessary to categorize wafers after boron deposition and prior to base diffusion by measuring their voltage-current (V-I) characteristics. The reason for doing this is that each V-I range requires a different base-diffusion time, and an over-diffused wafer is a lost wafer. A system has been put together using a microprocessor-based programmable calculator to do the following: take 5 V-I readings on the wafer; calculate the average and categorize it; present the operator with the category; maintain lot history; present histograms on demand; and record all historical data on a cassette at the end of a lot run.

The system has been operating for about nine months, and has increased first-run sales in the diffusion department by about 50%. In addition to the obvious savings, it would have been impractical to make these measurements by conventional methods because of their excessive labor content.



E. M. Mihalick
Solid State Division
Mountaintop, Pa.
Ext. 646

Communications and signal processing

Using a microprocessor as a coordinate converter

Test-range instrumentation equipment usually requires a means of obtaining real-time target acquisition information to and from a remote location. A data format using the earth-centered EFG coordinate system is in common use today. However, the instrumentation sensor must transform data from the EFG system to the local XYZ cartesian and Azimuth-Elevation-Range (AER) systems to be useful.

Most microprocessors are 4- or 8-bit machines and do not provide sufficient calculation accuracy to do this without lengthy software algorithms. Since the conversions should

be performed at a 10-pps rate, such a processor soon runs out of time. The Digital Equipment LSI-11 microprocessor, however, has enough accuracy and speed for this because it is a 16-bit machine with an extended arithmetic option that provides double-precision fixed-point and floating-point instruction. If the LSI-11 converter program is placed into a read-only memory, the system performs like a piece of hardwired equipment and needs no peripheral equipment.

D.J. Kleinschnitz
RCA Service Co.
Patrick AFB, Fla.

Spacecraft command and control system uses two COSMAC microprocessors

A unified low-power command and telemetry system using RCA CDP1802 microprocessors has been developed at RCA Astro-Electronics. The system's microprocessor-based central control unit interacts with two subsystems—the command decoder, itself microprocessor-based, and the remote distribution units (RDUs).

A transformer-coupled data bus carries supervisory command and telemetry words to the RDUs, which then return telemetry data to the central control unit for output buffering and final formatting. The command decoder takes signals from the spacecraft receiver and performs message

inspections and parity checks on them. The subsystem, which consists of less than one board of logic, decodes a set of 256 critical commands and directly distributes them to perform mission-critical command operations. All other properly formatted uplink data are transferred to the central control unit; the command decoder informs the ground station of improperly received data. The central control unit can perform both real-time and delayed-time processing. It also provides fixed and programmable telemetry format options, plus several rate options and telemetry storage features.

Charles Staloff
RCA Astro-Electronics
Hightstown, N.J.
Ext. 2256

Microprocessors in radar systems

Current applications of microprocessors to radar systems have centered on bit-slice bipolar microprocessors; two functions are considered prime candidates for microprocessor implementation in a phased-array radar system.

Coordinate-conversion is currently being implemented for a phased-array radar using the Advanced Micro Devices 2901 bit-slice microprocessor in a 24-bit configuration. This processor interfaces with a three-angle attitude reference system and performs the matrix calculations required to translate the moving antenna coordinates to stable coordinates. The result is a 50% processing load reduction

for one of the main system central processing units, making vital computation capacity available for other functions.

A beam-steering controller is also being implemented using a 16-bit processor to compute the phase tapers required to point the array. Other radar microprocessor applications under consideration include signal-processing functions, such as interpolation and monopulse implementation, and range and angle servo processors for tracking radars.

R.J. Smith
Missile and Surface Radar
Moorestown, N.J.
Ext. PM-2703

Low-power "micro-signal processor" made for smart sensor application

A signal processor based on the 8-bit COSMAC CDP1802 has been developed and brassboarded for sensor application. The processor's multiplier-accumulator circuit provides the micro with the substantially improved computational capability required to perform the various processing functions associated with "smart" sensors. The signal processor can automatically and periodically sample the analog outputs from seismic and/or acoustic transducers and then perform feature extraction and a

variety of pattern-recognition algorithms suitable for target identification, discrimination and classification. Feature extraction is currently based on frequency-domain analysis using the FFT algorithm. Computations are done with either 16-bit fixed-point or 24-bit floating-point accuracy. Specifications for the processor include the ability to perform, for example, a 32-point FFT within 20 ms, with power dissipation under 100 mW.

K. Prost, G. Dooley, and D. Hampel
Government Communications Systems
Somerville, N.J.
Ext. 6100

Microprocessors in telecommunications

M.D. Lippman

With microprocessors, telecommunications engineers can offer their customers new levels of cost/performance, increased reliability, and functions that were previously impossible or extremely costly to implement.

Michael D. Lippman joined RCA Laboratories in July 1966 as a member of the Research Training Program. In 1967 he joined the Communications Research Laboratory where he initially performed research aimed at developing efficient coding techniques for digital image storage and transmission. Since 1972 he has been working on microprocessor applications in data communications and broadcast equipment.

Contact him at:
Communications Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 2955



Microprocessors are appearing in virtually every type of telecommunications equipment. In telephone switching systems, intelligent terminals, data network controllers, multiplexers, communications test equipment, and other applications, the microprocessor is such a fundamental and pervasive tool that it is markedly affecting the basic architecture of telecommunications systems.

This article explores some of the benefits of microprocessors as they relate specifically to telecommunications. Also, the effects that microprocessors are having on the evolution of system architecture are discussed. Examples of microprocessor-based telecommunications equipment are given to demonstrate the advantages to be gained, to illustrate the effects on architectural development, and to give the reader an appreciation for the variety of applications that already exist.

Cost

LSI reduces cost-per-function.

The telecommunications industry has been turning to digital technology to meet an explosive increase in demand for services and equipment. This growing reliance is due primarily to the plummeting cost-per-function of digital circuitry over the last decade, compared with a modest decrease for analog circuits. Through increasing levels of integration, the functional complexity of digital integrated circuits has doubled each year while costs have stayed relatively constant.

The microprocessor lowers cost-per-function even further than custom LSI.*

*Large-Scale Integration.

Final manuscript received August 9, 1976.

This is so because the microprocessor is general and can be used to perform different tasks, whereas custom LSI devices are specific and require high production volumes before they are cost-effective. Therefore, many systems that could previously use only moderate levels of integration—the MSI** circuits used to implement hardwired logic—can now realize the cost advantage of LSI via the microprocessor.

LSI circuits lower more than just component costs.

The smaller size and reduced power requirements of an LSI implementation, as compared to discrete logic, can greatly reduce power supply and packaging costs. In addition, fewer interconnections yield increased reliability, and this means lower maintenance costs. This is a very important consideration in telecommunications systems where equipment is often geographically remote and sometimes inaccessible. It is the combination of reduced component, system, and support costs that makes the LSI circuit so economically attractive. And, although discrete logic will always be most cost-effective for some applications, the microprocessor is extending the total cost advantage of LSI to many medium, and even low-volume, systems and devices.

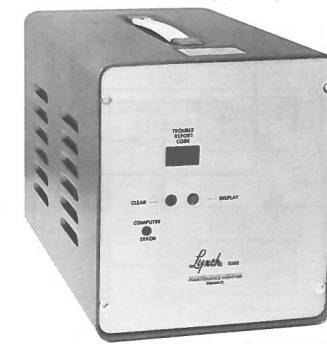
Intelligent telephones are coming.

An excellent example of an application in which low cost and small size are of great importance is AT&T's Transaction™ telephone. This is a special telephone set that functions as a remote terminal for credit authorization. It features automatic dialing, magnetic card and touch-tone pad data entry, and automatic operator

**Medium-Scale Integration.



Multiprocessor structure. Codex 6030 Series Intelligent Network Processor is built around an Intel 3000, controlling up to eight Motorola 6800S. It performs high level network management functions.



Monitoring and diagnosis. This B280 ESSS Maintenance Monitor by Lynch Communications Systems Inc. uses a microprocessor to isolate faults down to the printed-circuit-board level.



Flexibility. This GTD-120 stored-program PABX from GTE Automatic Electric is a good example of how the microprocessor can be used when a minicomputer cannot.

prompting. A Rockwell PPS-4 five-chip microcomputer is employed. This is a forerunner of what is almost certain to come: solid-state, "intelligent" telephone sets.

Flexibility

Telecommunications systems are constantly changing.

The programmability that makes the microprocessor a general-purpose (rather than specific) LSI device also allows microprocessor-based equipment to be reconfigured quickly and easily. Functional changes are made by changing the program, most often done simply by replacing a ROM. Thus modifications can be readily accomplished in the field, in sharp contrast to the rewiring required to change discrete logic, or the complete redesign needed with custom LSI. This flexibility is especially important in telecommunications, where systems must be very dynamic, expanding to meet new requirements and changing to accommodate new and better services, operating protocols, and hardware devices.

With microprocessors, flexibility is no longer a luxury.

To achieve this flexibility, minicomputers have been widely used in telecommunications applications. However, there are many areas in which the cost of minicomputers and their associated peripherals cannot be justified. In these applications, flexibility was previously a luxury that could not be afforded. But just as the microprocessor extends the economic benefits of LSI to lower volumes, it also extends the flexibility inherent in

stored-program control to smaller, simpler systems and devices.

The M1308 Multitran multiplexer by Computer Transmission Corp. illustrates the flexibility gained with microprocessor design. This device, designed for point-to-point multiplexing between terminal clusters and central computers, can handle many different kinds of communications channels. By changing the program, the user can completely rearrange the channel-scanning and data-handling sequence—an extremely costly, if not impossible, feat for random logic.

Digital private automatic branch exchange (PABX) telephone switching systems are good examples of how the microprocessor can be used when a minicomputer can't. The addition of stored program control to PABXs allows great flexibility in assigning lines and sophisticated functions such as call forwarding, camp-on,* and multistation hunt groups.** In large systems, the cost of the common control is shared by hundreds of lines, and the cost per line is low. In this case, a minicomputer is cost-effective. In small switches with one hundred or fewer lines, the cost per line of minicomputer common control is too high. Therefore, until the microprocessor arrived there were few, if any, small stored-program PABXs. Now there are several, including the GTD-120 120-line, 28-trunk PABX from GTE Automatic Electric.

*In camp-on operation, a call placed to a busy number will be retried automatically until it can be completed.

**When a call is placed to a busy extension that is part of a multistation hunt group, other numbers in that group will be successively tried.

With the low-cost microprocessor, even very small telephone electronic switching systems, along the lines of the Call Director, are feasible, offering more flexibility, more functions, and much lower cabling costs than electromechanical, button-per-line devices.

New functions

The microprocessor has made new functions possible for telecommunications equipment.

The microprocessor can implement functions that are difficult or impossible to implement with random logic. Many examples exist among current telecommunications devices. The Transaction™ telephone described earlier would certainly not be feasible if implemented with discrete logic. The intelligence that is routinely built into terminals today was very difficult to achieve before the microprocessor. Intelligence is now being added to a wide variety of test equipment resulting in greatly expanded functional capabilities. For example, Hewlett-Packard's Selective Level Measurement Set can make unattended measurements of FDM transmission parameters. The computational capability of the Intel 8008 microprocessor around which this instrument is built allows automatic analysis and data reduction. Rhode & Schwarz makes a Radiotelephone Test Assembly that, among other functions, corrects operator errors. Self-calibrating communications test equipment is being developed using microprocessors. New functional capabilities, made feasible by the microprocessor, improve the accuracy and simplify the operation of a wide range of telecommunications equipment.

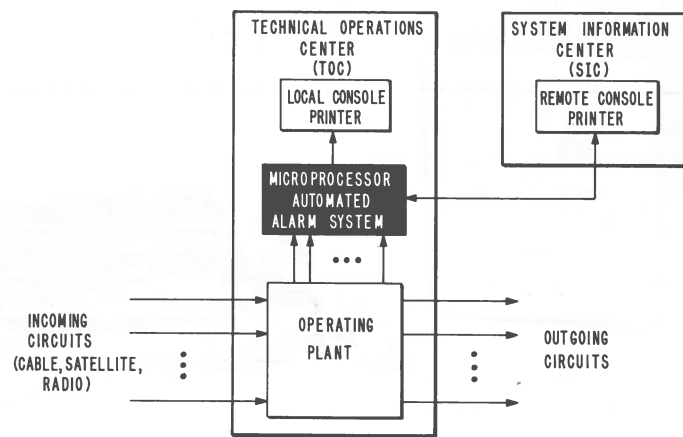


Fig. 1
Microprocessor Automated Alarm System is used by RCA Global Communications, Inc., to monitor various communications channels. The RCA COSMAC microprocessor provides control for the system.

The microprocessor is being applied to the vital function of fault monitoring and diagnosis.

Built-in diagnostic capabilities have long been overlooked for telecommunications equipment because they were so costly and difficult to implement. Microprocessor-based diagnostic systems are beginning to appear on the telecommunications scene. An example is the B280 Maintenance Monitor built by Lynch Communications to isolate faults down to PC-board level in its B280 electronic subscriber telephone switching system.

Another fault-monitoring system of this type is the Microprocessor Automated Alarm System (Fig. 1) developed by RCA

Global Communications. Built around the RCA COSMAC microprocessor, this system monitors the status of a variety of communications channels including trans-oceanic cables, satellite circuits, and radio links. The system, located at RCA Globcom's Technical Operations Center (TOC), generates line status reports on two console printers. A local console in the TOC receives failure and restoral reports (Fig. 2) for specific outages as they occur. A remote printer at a Systems Information Center (SIC) can request summary failure reports (Fig. 3) showing which lines are currently out of service. The current stand-alone system will eventually be tied into a microprocessor-based centralized monitoring system. It is almost certain that this

FAILURE:	NY-LD TDM1	05/01 12:13
RESTORED:	NY-LD TDM1	05/01 12:15
FAILURE:	NY-SF TDM1	05/01 12:24
FAILURE:	NY-SJ TDM1	05/01 12:24
RESTORED:	NY-SF TDM1	05/01 12:26
RESTORED:	NY-SJ TDM1	05/01 12:26
FAILURE:	NY-SF TDM1	05/01 13:02
RESTORED:	NY-SF TDM1	05/01 13:04
FAILURE:	NY-SF TDM1	05/01 13:07
FAILURE:	NY-SJ TDM1	05/01 13:07
RESTORED:	NY-SF TDM1	05/01 13:09
RESTORED:	NY-SJ TDM1	05/01 13:09
FAILURE:	NY-SJ TDM1	05/01 13:10
FAILURE:	NY-SF TDM1	05/01 13:10
RESTORED:	NY-SF TDM1	05/01 13:12
RESTORED:	NY-SJ TDM1	05/01 13:12
FAILURE:	NY-SF TDM1	05/01 13:25
FAILURE:	NY-SJ TDM1	05/01 13:25
RESTORED:	NY-SF TDM1	05/01 13:27
RESTORED:	NY-SJ TDM1	05/01 13:27
FAILURE:	NY-SF TDM1	05/01 13:28
RESTORED:	NY-SF TDM1	05/01 13:30
FAILURE:	NY-SF TDM1	05/01 13:31
FAILURE:	NY-SJ TDM1	05/01 13:32
FAILURE:	TTT CONVERTER	05/01 13:33
RESTORED:	NY-SF TDM1	05/01 13:34
RESTORED:	NY-SJ TDM1	05/01 13:34
RESTORED:	TTT CONVERTER	05/01 13:38

Fig. 2
Line status reports for the alarm system shown in Fig. 1 identify specific line failures and restorals as they occur.

FAILURE REPORT 11/06 14:30	
TAT-1	
TAT-2	
TAT-3	
TAT-4	
TAT-5	
CANTAT-2	
FST-2	
INTELSAT 1VF3	
INTELSAT 1VF7	
BERMUDA CABLE	
STCM OK-VF 1201	
STCM OK-VF 1202	
STCM OK-VF 1203	
STCM OK-VF 1204	
STCM OK-VF 1205	
SATCOM SYSTEM	
CTE	
ARC	
CTS	
AIRCON	
TTT CONVERTER	
NY-LD TDM5	
NY-LD TDM3	
NY-SF TDM1	

Fig. 3
Summary failure report shows which lines are out of service at any given time.

type of diagnostic and monitoring capability will be built into most large telecommunications systems in the near future.

Systems architecture

The trend towards distributed architectures is very strong in telecommunications because these systems are inherently distributed, both geographically and functionally.

The microprocessor has begun to influence the basic structure of many types of electronic systems. By offering very small, low-cost increments of processing power, the microprocessor allows the workload to be distributed among various components in a system. While the minicomputer started this trend in large systems, it has taken the microprocessor to extend the range of this architectural approach to smaller, simpler systems and devices. A classic example of the type of structures that are evolving is the use of smart terminals, remote devices that have processing capabilities and can function to some extent independently of the central processing facility. This arrangement offers several advantages. First, some routine tasks and pre-processing can be removed from the central processor, freeing it for more important work. Second, if the central processor or communications link becomes inoperative, the terminal can still function, even if only in a limited way. Thus, for example, remote data collection need not stop just because the central computer is down.

Front-end processors represent another important application.

Another important distributed-system application for microprocessors is beginning to emerge: front-end communications processors for data networks. The front-end processor, usually implemented by a minicomputer, is designed to be connected between the central processor and its remote terminals (see Fig. 4). It performs most of the tasks associated with terminal communications, such as character and block assembly, error control, and line supervision, thereby freeing the main processor for more applications-oriented jobs. The front-end processor also assumes the burden of interfacing the specific terminals and devices, and presents a single, standard peripheral interface to the main processor.

The front-end processor must therefore be able to accommodate a wide variety of terminal types. In addition, it must be able to handle different communications

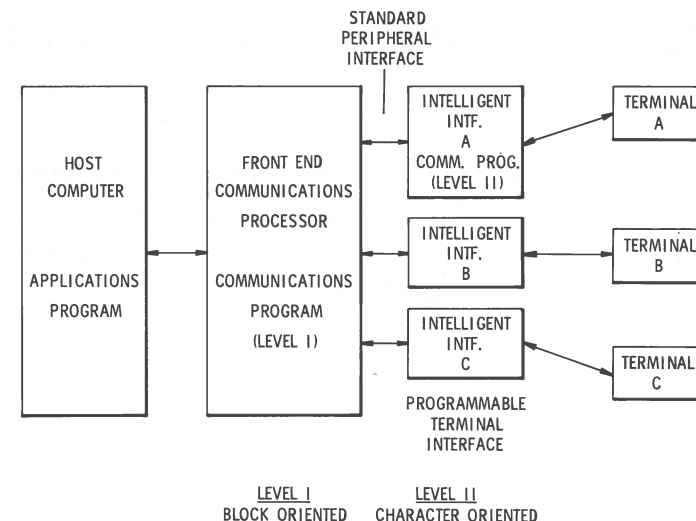


Fig. 4
Front-end communications processor provides a connection between remote terminals and a central processor.

protocols and control procedures. In the past, this flexibility has been provided by a family of line interface units and custom software. Recently, however, microprocessors have been incorporated in intelligent line interfaces (Fig. 5), which are able to accommodate different terminals and protocols in software. Thus, only a single hardware module is needed. In addition, line interface units can ease the burden on the front-end processor.

For example, the microprocessor can easily perform the character-by-character processing on its dedicated line. The front-end processor then performs only block processing instead of the character-by-character processing for the aggregate of all lines. Thus, processor throughput requirements are substantially reduced. The recently announced DMC-11 network link from Digital Equipment Corporation is an example of such a device. It performs all of the functions of DEC's DDCMP line

protocol including message assembly, header processing, and error checking. This programmable line interface approach can be extended to microprocessor-based format translators that convert messages from diverse input devices into a standard format for processing.

Multiple-processor structures have several advantages.

The multiprocessor network is another distributed architecture in which the microprocessor is finding application. Such a structure comprises several processors each performing a part of the total task. For example, several new terminals have come on the market recently, using two microprocessors, one for editing functions and one for communications functions.

A good example of the benefits of multiprocessor configurations is the Model

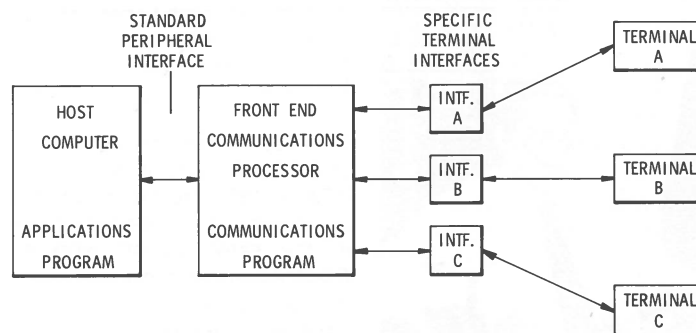


Fig. 5
Line-interface units can perform character-by-character processing (level II), freeing the front-end processor for block processing (level I).

6000 Intelligent Network Processor from Codex. This device, built around an Intel 3000 controlling up to eight Motorola 6800s, performs high-level network management functions such as automatic channel assignments, error checking, and system diagnostics. It can dynamically reconfigure a network to bypass faulty or busy links. The system is reputed to have the processing power of two medium-sized minicomputers.

Multiple-processor structures have several other advantages. They provide a high degree of modularity since functions can be added independently. They increase flexibility because it is easier to change individual functions once they are isolated. It is also possible that, in the future, different microprocessors will be optimized for different functions (e.g., bit manipulation for communications, or BCD arithmetic for calculations) which will make the multiprocessor approach even more attractive.

Conclusion

Only with extremely cost-effective equipment can the telecommunications industry sustain, within the limit of available capital, the phenomenal growth rate that is being dictated by customer demand. Currently available microprocessor devices have already proved to be flexible and economical systems building blocks. Microprocessors are already being widely used to replace random logic, as lower cost alternatives to minicomputers, and to provide entirely new functions and devices.

In spite of the large number of microprocessor applications that already exist in telecommunications, what we have seen up to now is only the beginning of the role the microprocessor will eventually play. As devices that are cheaper and more powerful emerge, entirely new areas of application will be opened up to programmable designs. For example, one of the next steps in the evolution of LSI will be the microcomputer on a chip: CPU, RAM, ROM, and I/O all in a single IC. Simple devices like this are already available, and newer, more-sophisticated ones have been announced. The potential of such devices for all areas of electronics is truly astounding.

References

1. Garen, E.R.; and Lazar, L.; "Microprocessors in telecommunications," *Telecommunications*, (Apr 1976), pp. 43-48.
2. Kaye, D.N.; "Microprocessors help to communicate by voice and bit stream," *Electronic Design* (Apr 12, 1976), pp. 58-61.

Microprocessors in manufacturing and control

C.C. Wang
C.T. Wu
P.M. Russo
A. Abramovich
A.R. Marcantonio

The dedicated microprocessor may foster a "second industrial revolution." Three reasons for this are the standard interface bus, EPROM software development, and integer-arithmetic computation.

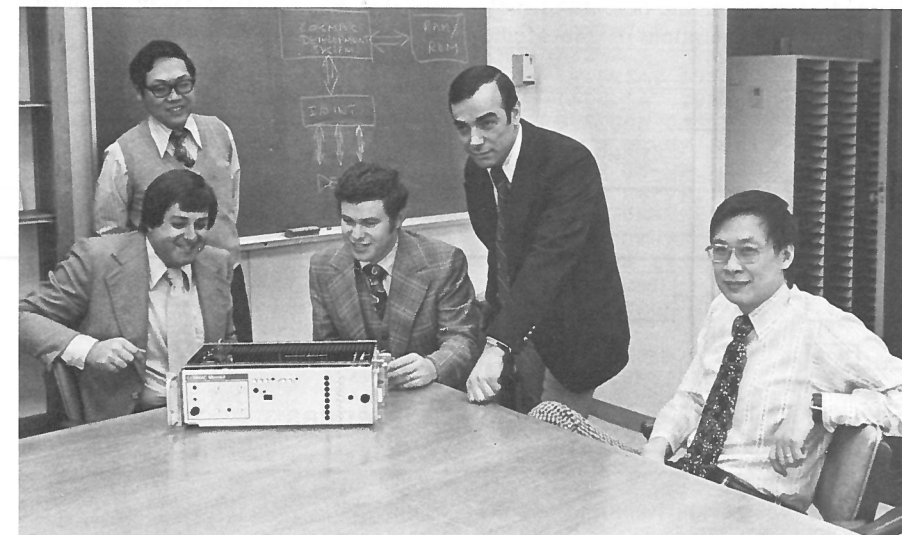
As computers become smaller and less costly, they no longer play the role of superstars waited on by rooms full of peripherals. On the contrary, machines and equipment are becoming centers of attention catered to by microprocessors. This decentralization of computing power has been a natural consequence of cost drops in semiconductor components, coupled with rising prices for mechanical subsystems.

In industrial applications, microprocessors hold the promise for what some enthusiasts call "the second industrial revolution." Intelligent machinery with plenty of muscle and brain power should be the rule, rather than the exception, in the future. Instead of being limited to the classical role of computations on abstract quantities, microcomputers are at home with real attributes such as voltages, temperatures,

Final manuscript received December 22, 1976.

The biographies of Paul Russo and Angelo Marcantonio appear with their other papers in this issue.

Authors Wang, Russo, Abramovich, Marcantonio, and Wu.



pressures, etc. In this context, truly innovative uses of computers are just beginning to be explored.

This paper reviews some of the applications of microprocessors in this area and also describes some of the considerations associated with the design of such applications.

Computer-aided manufacturing

In the electronics industry, the pressures of increasing labor costs and the need for better quality control pushed the need for computer aids to manufacturing (CAM) to the surface. Microprocessors are making impressive gains in this field because they can provide solutions to long-existing problems in the factory. Although automation, up to now, has achieved its biggest successes in heavy industries such as steel-making, oil-refining and chemical-

processing, all aspects of manufacturing are moving toward more automation.

The manufacturing process can be roughly divided into five phases, as far as electronics use is concerned. They are:

- design
- documentation
- fabrication and assembly
- testing
- stocking and distribution

Abe Abramovich joined RCA in 1976, and is currently involved in exploring microprocessor applications. He is also attending the Graduate School of Engineering and Applied Science at Columbia University.

Contact him at:
Systems Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 2750

Chih-chung Wang was a senior scientist with B.F. Goodrich Co. before he joined RCA Laboratories in 1973. Dr. Wang has been working in computer-aided design, software development for integrated-circuit testers, and, more recently, microprocessors and microprocessor applications.

Contact him at:
Systems Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 3325

Chin Tao Wu has worked on various aspects of computer systems design since joining RCA in 1965. He has been interested in microprocessors since 1971 and was active in the early support work for RCA's COSMAC microprocessor.

Contact him at:
LSI System Design
Solid State Technology Center
Somerville, N.J.
Ext. 6061

Fig. 1 illustrates the process flow in such a manufacturing environment. Computers have been integrated into the different phases of this scheme with varying degrees of success. For design, documentation, and stocking and distribution, large main-frames and minicomputers are being used most extensively. In the area of testing, minicomputer CPUs (Central Processing Units) often control the switch-sequencing for high-speed testers. The microprocessor is, of course, a more cost-effective solution in low-speed testing.

In fabrication and assembly, tasks are often tedious, repetitive, and demanding of operator concentration. Microprocessors taking over these burdens will have the greatest impact on the eventual automation of this phase. Robots equipped with "eyesight" are already sitting on some of our assembly lines.

Wafer fabrication in the semiconductor industry is an example showing how better monitoring and control of processing steps can definitely pay off. In wafer-fab, many cycles of masking, etching and diffusion are required. Any deviation in the sequence means that subsequent steps are completely wasted. The penalties in lost time and poor yield are severe.

Precise process-monitoring can prevent a bad lot from propagating through a line. More accurate control, on the other hand, can minimize the occurrence of these lots. Both functions are within the realm of microprocessors.

Microprocessors are now being used in process monitoring and control for semiconductor manufacturing.

In the past, various industries have successfully applied minicomputers in controlling product flow. In the RCA Solid State Division, a system for collecting product flow data called OPEN (On-line Production Entry Network) has been implemented and is operating in several product lines. The OPEN system, however, collects logistical data (i.e., where products are at any time) rather than the process information giving rise to these products. Recognizing the need for collecting critical process parameters themselves, an ambitious process monitoring and control program (PMC) has begun.

The PMC system (Fig. 2) is based on a Data General Eclipse minicomputer and a

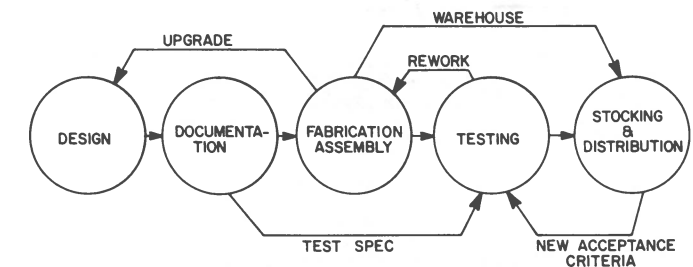


Fig. 1
Computer-aided manufacturing can be beneficial at any point in the manufacturing process. Large main-frames and minicomputers are already working extensively in design, documentation, and stocking and distribution; microcomputers are becoming more evident in the fabrication and testing areas.

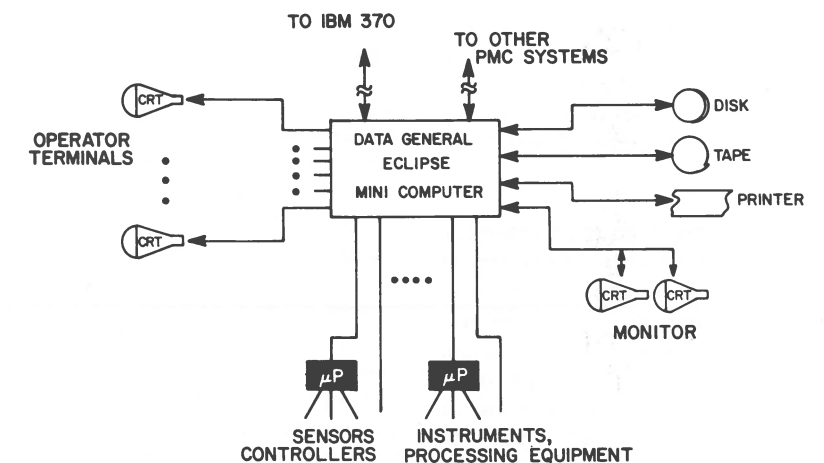


Fig. 2
"Local intelligence"—microprocessor controllers with local-loop capabilities—and a central minicomputer make up the process monitoring and control system.

network of CRT terminals similar in configuration to the OPEN system. Intelligent instruments and controllers with local-loop capabilities perform the measurement and control functions in order to minimize the effect of host-system crashes. Microprocessors form the basis for local intelligence in this level of the distributed system.

Microprocessors are the brains of the intelligent instruments that measure process variables and communicate them to a central data base, where the information is sorted and analyzed for statistical control purposes. A COSMAC-based capacitance voltage monitor² is an example of such a satellite instrument that, in addition to communicating the information, frees the operator from cumbersome calculations.

While process-monitoring provides the necessary feedback for corrective actions where man is in the loop, automated feedback control of complex processes is

also important. Semiconductor-equipment vendors have designed closed-loop control into machines such as epi-reactors and thin-film depositors by using microprocessors. When linked with a host computer as components in a distributed system (such as PMC), these equipments acquire the additional flexibility of changing their control functions on request. With the rapid pace of technological obsolescence in the semiconductor industry, this capability is an important asset, since the same equipment can then perform many variations of a basic process.

Automatic testing had been confined to complex systems, but microprocessors are making it economical for subsystems and even components.

Automatic Test Systems (ATS), which can be used to automatically identify component, subsystem and system faults, are becoming increasingly important from the view of reliability and maintainability. Reliability is improved with ATS since marginal components and subsystems are

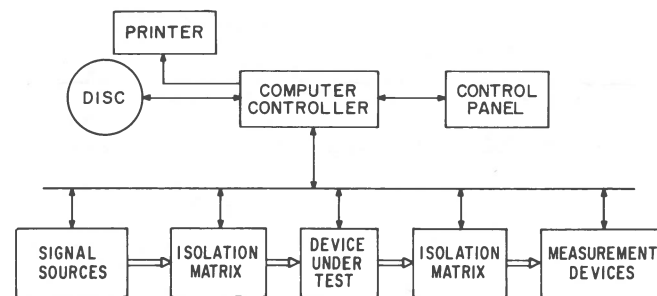


Fig. 3
Typical automatic tester includes signal package, measurement package, and a processor controller. I/O, data-storage, and operator-control subsystems complete the unit.

replaced prior to equipment shipment, resulting in a lower probability of field failure. Ease of maintenance is also improved since system-level tests can identify the probable subsystem failures, and so give rapid repair turnaround and hence reduced downtime.

The principal subsystems of an automatic tester are a signal package, a measurement package, and a processor acting as system controller. Additionally, some operator control is needed along with some output devices (displays, meters, printers) to identify the test results. Finally, a disc or other mass-storage system may be desirable for storing data for later statistical analysis. Fig. 3 presents a block diagram of a typical test system where the subsystem under test is isolated from the signal sources and measurement devices. These can be relays, solid-state switches, or simple resistors, as required. Note that the system of Fig. 3 is rather complete—not every test system will contain all of its indicated subsystems.

ATS has historically been used in large military applications and in automatic testing of complex parts, where expensive test fixtures can be economically justified. The advent of microprocessors has opened many avenues for low-cost dedicated testers, which means that industries can now economically justify automated and more complete testing of low-cost/low-complexity subsystems right on down to the component level.

Current economic factors are tending to accelerate the swing toward more fully automated testers. Ever-increasing labor costs, decreasing hardware costs, and demands for constantly-improved product reliability and safety all dictate increased automation. Additionally, it is becoming increasingly desirable to catch failures early in the assembly process, since the value

added (cost to troubleshoot and repair) goes up exponentially with the number of components in the subsystem under test.³

Many types of testing are possible. These include functional, dc, and ac (dynamic) tests. Functional tests establish that the subsystem under test is performing a desired digital or analog function. For example, a typical digital functional test would be to determine if a logic function satisfies a specified truth table, and an analog functional test would be to see if an amplifier meets specified linearity limits over a prescribed input-signal range. DC tests usually imply static measurements of component values, PC board continuity, amplifier gain, saturation voltages, etc., and ascertain whether these satisfy a given set of specifications. Dynamic tests establish relevant time-domain parameters such as system delay, overshoot, etc., and may also be used to measure reactive components.

The test methodology for a system defines the broad strategy of what will be tested and how. Designers must give the methodology considerable thought, for no matter how well the actual tester is performing, it will be of little use if it fails to test the critical components or critical functions.

Once a test method has been chosen and the test system developed, the test specifications must be defined. This step is crucial and very complex since it involves subtle trade-offs between performance/reliability objectives and system cost. Too-rigid test limits will increase system cost (more rejects), and too-liberal test limits may lower the average system performance and, perhaps, reduce reliability.

From this we conclude that both test methodology and test specifications are strongly influenced by economic and

marketing factors, which will dictate the trade-offs of optimal cost per test vs. system performance and reliability.

If you are looking for more information on automated testing, Refs. 3 and 4 present excellent overviews of its desirability and economic benefits. Ref. 5 describes a host of testers having microprocessors as their controllers, and Ref. 6 describes a COSMAC-based tester-oriented development system described in depth elsewhere in this issue.

Dedicated controllers are replacing main-frame computers.

Before the use of microprocessors became widespread, mini or large computers were the hearts of complex controllers. Because of this centralization, controllers were economically practical only for large and complex systems. Furthermore, these early systems had all the attendant disadvantages of centralized control—large cabling costs, decreased reliability, and less flexibility. Smaller systems were designed with relay logic, sequencers, or, perhaps, analog feedback control.

With the advent of low-cost microprocessors, it became possible to decentralize the control function and use intelligent controllers dedicated to specific tasks. Many advantages result from local-loop and distributed control, including lower cabling costs, reduced noise pickup, improved overall reliability (if one small system fails, the remainder keep operating), simpler maintenance, and more flexibility. Several brief examples of dedicated microprocessor-based controllers follow. Refs. 7-11 discuss the benefits and applications of microprocessors to industrial control in more depth.

One example is in using a microprocessor to control the speed of a hysteresis synchronous motor to a high degree of precision. These motors maintain accurate average speeds, but their instantaneous speed may vary considerably. A microprocessor may be used to monitor the motor speed (say by counting crystal-controlled clock pulses per revolution) and then issue signals to move the driving frequency in the right direction, Fig. 4. With this technique, an all-digital motor-speed-control system is capable of speed accuracy of one part in 10^6 .

A second example, microprocessor-based automotive spark-timing control, has the

motivation of improved engine performance, fuel economy, and exhaust emission. The speed of combustion of the air fuel mixture is a strong function of engine operating conditions (rpm, load, etc.). Hence, it is important to have variable spark timing since it initiates the combustion process.

A.D. Robbi and J.W. Tuska have developed such a COSMAC-based timing system.¹² The system (Fig. 5) senses the vacuum advance, crankshaft angle, rpm, and dwell, and then triggers a standard electronic ignition subsystem. The CPU adaptively computes, for each spark, the desired firing angle (spark timing).

Other microprocessor control applications abound. Microprocessors are currently being designed into communications controllers (Harris 8400 multiplexer, Codex 6030 intelligent network processor), numerical-control machine tools, tracking systems, and instruments (HP 1722A scope, Systrom-Donner 7115 digital multimeter, DANA 9000 counter/timer). It is apparent that we see only the tip of the iceberg!

Design considerations

The design of a microprocessor-based application consists of selecting the central processing unit (CPU), memory, and I/O devices, and then developing customized I/O hardware, I/O interface, and software. Instead of presenting an extensive overview on the subject, only a few selected topics are discussed in detail here—specifically, the IEEE 488 Interface Bus, EPROMs and software development, and integer arithmetic. Ref. 13 gives an overview of system design.

The I/O interface design comprises a major portion of the hardware development for most of the microprocessor-based applications. Using the IEEE 488 Standard Interface Bus cuts back on this development work, since a wide variety of standard instruments can be interconnected quickly and easily with it.

An EPROM (Erasable Programmable Read Only Memory) is a very flexible medium for program storage. It is very popular in prototyping and low-volume applications, and it provides an ideal medium to integrate software into the application hardware.

Since most microprocessors perform multiplication and division with time-

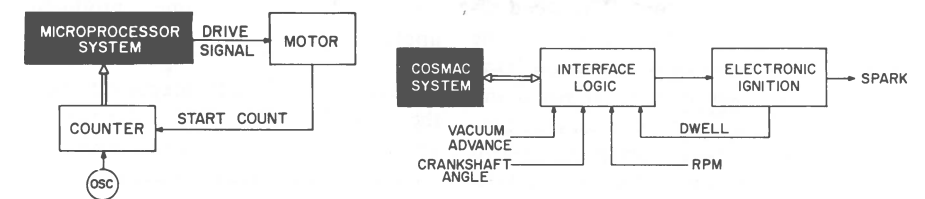


Fig. 4
Motor controller is an example of dedicated control. All-digital system is capable of one-part-in-a-million speed accuracy.

Fig. 5
Dedicated COSMAC microcomputer controls automotive spark timing more accurately than the mechanical system it replaces.

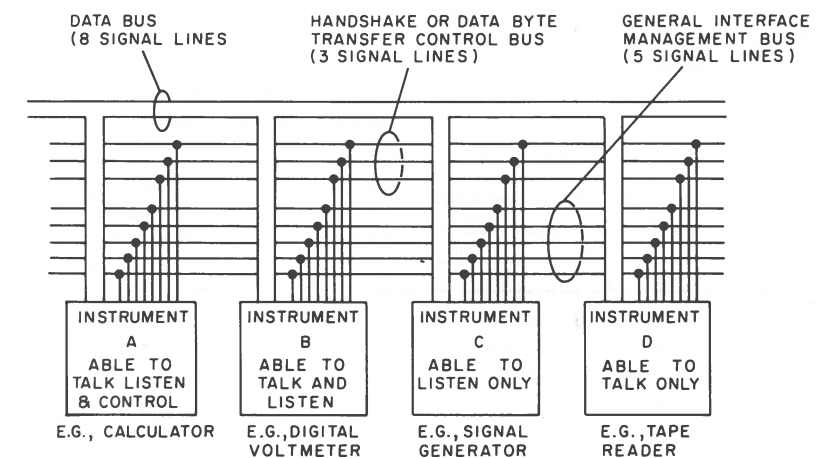


Fig. 6
Standard interface, the IEEE 488 bus, allows instruments to talk to and control each other. Sixteen signal lines are divided into groups for data I/O, "handshaking" for data transfer between devices, and controlling.

consuming software subroutines, microprocessor applications have been limited in the control area. Integer arithmetic is a technique that avoids multiplication and large intermediate results in control algorithms; it is especially suitable for microprocessor applications.

The standard interface bus lets instruments talk to, listen to, and control each other.

The IEEE 488 Interface Bus provides a versatile and effective communication link^{14,15} for exchanging digital information in an unambiguous manner. It can accommodate a wide range of devices with similar protocols and is being proposed as an international standard—the number of instruments supported by the bus grows almost daily.

The interface bus consists of a group of 16 signal lines (Fig. 6) interconnecting a number of devices. Any device connected

to the bus may either be idle or functioning as a talker, listener, or controller. As a talker, a device can send digital information across the bus to any device acting as a listener. Conversely, listeners can receive digital information across the bus from talkers. A controller can address other devices as talker or listener, act on service requests, and issue commands. A system controller can perform all the controller functions, remotely control other devices connected to the bus, and initialize the bus. At any given time, only one device can be functioning as the system controller.

The 16 signal lines that comprise the bus may be classified into three separate sets—the data bus, the handshake group, and the bus-management lines. The data bus consists of 8 bidirectional I/O data lines and the handshake group is made up of 3 lines that help synchronize data transfers between devices. The controller uses the 5 bus-management lines to process service

requests, sense and control the end-of-record line for data transfer, and set the attention line high or low to indicate if the data bus is in the address or command mode. The system controller uses the bus-management lines to remotely control other instruments on the bus and send the interface "clear" for a system reset.

If an instrument is being designed to interface to the IEEE 488 standard bus, the instrument designer must select which functions to include, as well as the capabilities within the chosen functions. In the event that an interface is being developed for a controller, and the controller has software-processing capabilities, it's up to the designer to decide what parts of the interface should be done in hardware and in software.

EPROMs simplify software development.

In developing microprocessor-based applications, the software-storage medium is an important subsystem. For sophisticated systems, we can depend on permanent program storage with paper tape, cassettes, floppy disks or even the storage media of a large computer. But, obviously, these methods cannot be used with most of the intended microprocessor applications. For most applications, ROMs (read-only memories) and EPROMs (erasable programmable read-only memories) provide attractive alternatives for nonvolatile program storage.*

ROMs are mask-programmable in the sense that the software is built into the IC chip. Because a mask must be designed to customize the chip for each piece of software, ROMs are only economical for large-volume applications. ROMs have the disadvantage that once manufactured, they cannot be modified. Also, it usually takes 6-8 weeks for delivery of a new ROM; this delay is intolerable in the product-developing and prototyping stage.

These problems can be avoided with EPROMs, which are based on the floating-gate MOS structure¹⁶—they store information in the memory cell as electrons trapped in the floating gate. The programming takes only a few minutes. Because EPROMs can also be erased and reprogrammed for program modification, they are an ideal storage medium in the product-developing and prototyping stage. They also have the economic edge over

*For certain applications, a battery-backed CMOS RAM, e.g. RCA CDP1821 (1024x1) or CDP1822 (256x4), may also be very desirable.

ROMs for low-volume production applications.

To appreciate the advantages offered by the EPROM, we have to understand the approach to software development for microprocessor-based applications. It consists of some or all of the following steps.

- 1) Software developed and debugged on a time-sharing system.
- 2) Software developed, tested and debugged on a microprocessor-based development system.
- 3) Software tested on the prototype.
- 4) Software integrated into the final product.

EPROMs can be used as the program storage medium in each of these steps except for the first, but the most important applications of EPROMs are in the last two steps. In the time-sharing and microprocessor-based approaches, EPROMs perform the same function—transferring software to the prototype and the final product.

Most of the microprocessor-based development systems provide certain basic system utilities, e.g. examining memory, modifying memory, setting breakpoint, etc., in order to generate, edit, and debug the software. These primitive tools, coupled with an EPROM's program storage ability, are adequate for certain application developments. At the beginning of each work session, the contents of the EPROM are loaded into the RAM of the development system for editing and testing, and at the end of the session, the contents of the RAM are dumped into the EPROM for storage. After the program is debugged, the EPROM can be used directly in the prototype or the final product.

Since this approach deals with everything at the machine-code level, its capability for system development is limited. It does, however, provide a simple method for upgrading or customizing the software for a microprocessor-based product that uses an EPROM for program storage.

Integer arithmetic solves control problems at high speed.

Integer arithmetic is a numerical technique for computing solutions to initial-value problems for functions $f(x, y, \dots) = 0$. Here, the term "initial-value problem" signifies that rather than solving the equation for an exact solution, an arbitrary starting point is chosen at a set of coordinates $x, y, \dots$. From

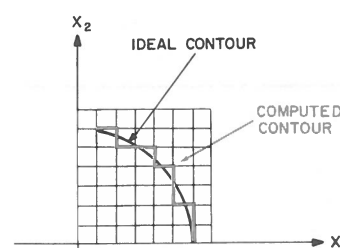


Fig. 7 Integer arithmetic solves equations to the closest grid point (which can be set as accurately as needed for a given control application).

that starting point, the integer-arithmetic technique converges onto the desired contour, and then follows it.

This technique is designed for digital computation, and it is especially suited for microprocessors because it avoids time-consuming multiplication routines and so does not require the storage of large products. In solving initial-value problems of the type $f(x, y) = 0$, a new variable, F , is introduced, and the equation is set to $f(x, y) = F$. The problem is then solved on a set of Cartesian coordinates only to the closest grid point (Fig. 7). Since most digital control applications require a finite accuracy, the grid is chosen so that its smallest increment is consistent with the system's tolerable error. The resulting error of the integer-arithmetic solution is predictable and noncumulative because F is evaluated at incremental integer points and expressed as a difference for each variable, holding the other variables constant. In a two-dimensional case, the incremental values are

$$F_{\pm x} = \pm 2x + 1$$

$$F_{\pm y} = \pm 2y + 1$$

The technique can solve any initial-value problem, linear or non-linear, of any order.

Fig. 8 shows one of the several computing algorithms that have been formulated for integer arithmetic. This one is designed to examine the magnitude or sign of $F|_{x,y}$ and then decide to make either a positive or native change in x or y in order to keep the F variable within the allowable error limits.

An Intel 8080A microprocessor has used this algorithm to solve for the set of points on a circle, given the initial x, y coordinates (Fig. 9); the introduction of a damping factor yielded the spiral contour. The program consisted of about 150 in-

structions, which yielded a new solution to the equation once every 600 microseconds. It would not be possible to execute the solution to this problem at comparable speed using the classical method.

This algorithm may be applied with equal ease to solve contours for other second-order equations, and the integer-arithmetic technique may be applied to control applications that require high-speed real-time solutions.

Conclusions

As the engineering community's familiarity with microprocessors increases, applications in manufacturing and control will undoubtedly become wider and deeper. Mass-produced microprocessors provide cheap and flexible substitutes for relay logic, sequencers, or perhaps analog feedback control. They are being designed into more and more machines, equipment, and instruments, providing local programmable intelligence and control. The spread of these intelligent machines over the manufacturing plant distributes the processing power (which used to be centralized in large main-frames) and lays the foundation for the distributed processing system in the future.

References

1. Cassell, D.A.; *Introduction to Computer-Aided Manufacturing in Electronics* (Wiley Interscience; 1972).
2. Wu, C.T.; "CVM: a microprocessor-based intelligent terminal," *this issue*.
3. Lyman, J.; "Automatic circuit testers lead the way out of continuity maze," *Electronics*, Aug 1974, pp. 87-95.
4. Eleccion, M.; "Automatic testing: quality raiser, dollar saver," *IEEE Spectrum*, Aug 1974, pp. 38-43.
5. Runyon, S.; "Microprocessors showing promise in test equipment, but haven't made it big yet," *Electronic Design*, Apr 26, 1974, pp. 90-95.
6. Marcantonio, A.R.; and Russo, P.M.; "A microcomputer for test and control applications," *this issue*.
7. Weissberger, A.J.; "Microprocessors simplify industrial control systems," *Electronic Design*, Oct 25, 1975.
8. Olson, K.; "The tractors of the computer industry—part 1," *Digital Design*, May 1975.
9. Olson, K.; "The tractors of the computer industry—part 2," *Digital Design*, Jun 1975.
10. Weissberger, A.J.; "Microprocessors expand industry applications of data acquisition," *Electronics*, Sep 5, 1974.
11. Marmala, A.D.; "Benefits of localized control with microcomputers," *Computer Design*, May 1975.
12. Robbi, A.D.; and Tuska, J.W.; "A microcomputer-controlled spark advance system," *this issue*.
13. Russo, P.M.; "Architectural trade-offs in designing microcomputer systems," *this issue*.
14. IEEE; "IEEE standard digital interface for programmable instrumentation," *IEEE Std.* 488-1975.
15. Ricci, D.W.; and Nelson, G.E.; "Standard instrument interface simplifies systems design," *Electronics*, Nov. 14, 1974, pp. 95-106.
16. Frohman-Bentchkowsky, D.; "A fully decoded 2048-bit electronically programmable FAMOS read only memory," *IEEE Solid-State Circuits*, Vol. SC-6, (Oct 1971) pp. 301-306.
17. Gorman, J.E.; and Raamot, J.; "Integer arithmetic technique for digital control computer," *Computer Design*, Vol. 19, No. 7, (Jul 1970) pp. 51-57.

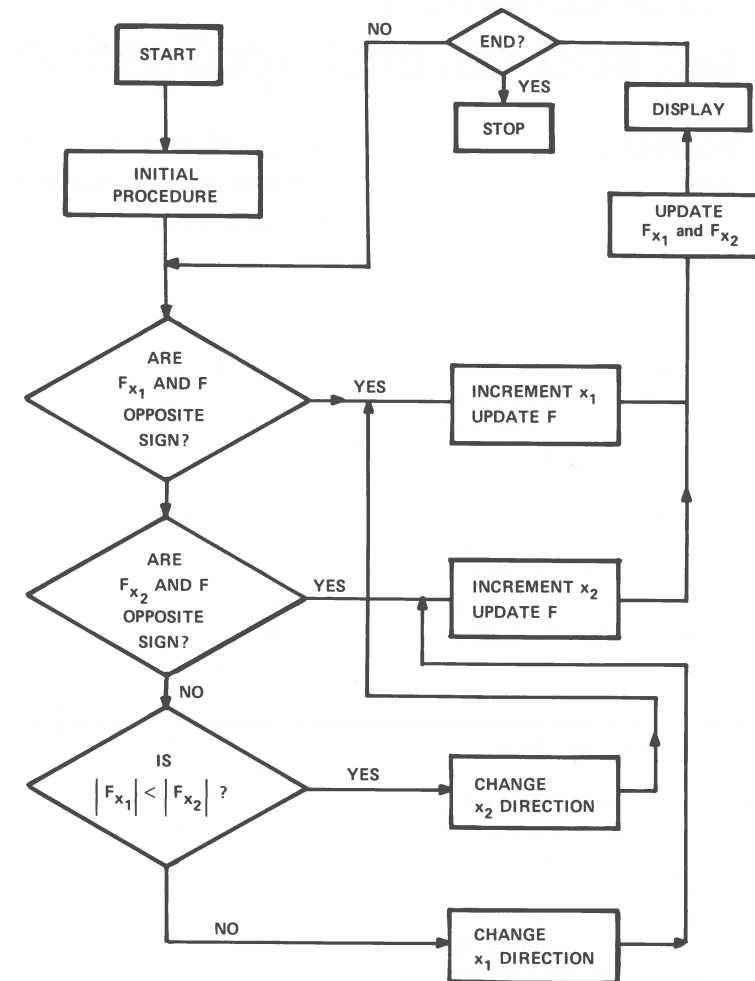


Fig. 8 Integer-arithmetic technique makes high-speed microprocessor solutions to equations possible by eliminating multiplication and storage problems. Algorithm shown here solves contours for second-order equations.

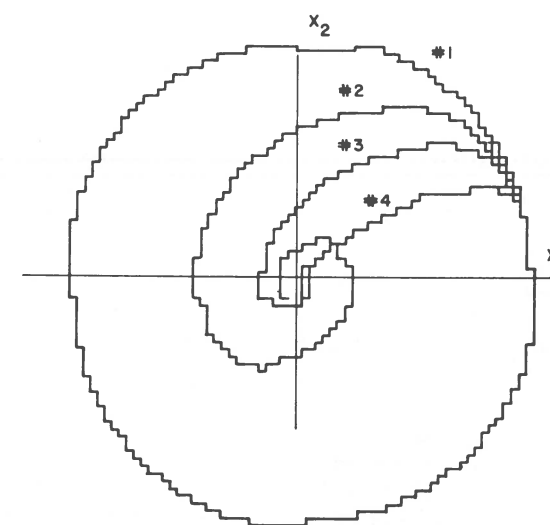


Fig. 9 Results of using the algorithm of Fig. 8 to follow the contours of a circle and spirals.

A microcomputer for test and control applications

A.R. Marcantonio
P.M. Russo

This versatile microcomputer system controls instruments as diverse as tv displays, programmable power supplies, timer/counters, and other microprocessors.

Low-cost microprocessors, memories, interface circuits, sensors, and transducers are now making it possible to decentralize test and control functions. With flexible, modular, economical, and expandable microprocessor-based hardware available, many test and control functions that can not economically justify a powerful minicomputer-based system may now be ready candidates for the low-cost microprocessor-based approach.

Advantages of microprocessor control

The primary advantage of computer-based testers and controllers is the ease with which they can be reprogrammed to satisfy a new test or control sequence using different parameter values. With modular hardware, the majority of functions can be provided by simply plugging in the required interfaces and modifying the software. The only custom circuitry required is used to condition the signals between the computer system and the device under test or control. This standardization yields tremendous benefits from the maintenance and backup-unit points of view; furthermore, it reduces the cost per tester, since the basic hardware may be reproduced many times over.

System organization

Fig. 1 details the components of a microcomputer test and control system designed at RCA Laboratories. The COSMAC Development System (CDS)¹, which is the center of the system, is equipped with 4K bytes of RAM and supports two-level I/O.²

In defining the device interface architectures for this system, the overall aim was to keep them both general and easy to use. No attempt was made to conserve I/O instructions, since two-level I/O makes the

entire I/O interface available to each device. Finally, I/O instructions can easily be incorporated into a software interpreter aimed at a particular application.

The following paragraphs give more information on the interfaces listed in Fig. 1.

The analog-to-digital converter reads and converts up to 16 input ports.

The analog-to-digital interface uses the 0-10 V 12-bit Datel ADC-M12B1B1 A/D converter with a 13- μ s conversion time and a 10-megohm input impedance. It allows an analog voltage to be read from any one of sixteen input ports (two of which are tied to stable reference sources) and converted to a binary value that can then be read by the CPU. The interface architecture allows for compatibility between 8-, 12-, or 16-bit converters without altering the interface logic design or the I/O instructions. The same data format is used in both the D/A and A/D converters to allow for compatible data-handling software.

Two of the program-selectable input ports are dedicated to stable voltage references—5V and ground—to permit a certain amount of self-checking and data correction via suitable software techniques. An erroneous digital value for the 5-V reference indicates an error in gain, offset, or both, while an erroneous digital value for the ground reference indicates a positive offset error. The gain error is the difference in slope between the actual transfer function of the A/D converter and the ideal function. Offset error is the amount by which the transfer function of the A/D fails to pass through the origin, referred to the analog axis.

Multiple-device interrupts are resolved quickly by hardware priority and software vectoring.

The hardware³ operates as follows: During each machine cycle, the status of 8 in-

terrupt lines is sampled and latched; the output of the latches drives a priority encoder, which can be read by the CPU as a normal memory location. The distinguishing feature of this method is that it accomplishes a direct jump to the interrupt service code of highest priority by stuffing the output of the priority encoder into the low-order byte of the program counter. The interrupt priority-resolution hardware features are summarized below:

- The highest-priority interrupt is always serviced first.
- Any of the eight devices may be inhibited from interrupting the CPU.
- Interrupt-resolution hardware appears as a memory location, so no I/O instructions are required for access and it is always selected.
- A jump to a respective device's service code can be done with two instructions.
- A single address accommodates both the priority encoder on read and the interrupt-enable latches on write.

The programmable counter/timer interface makes it possible to control an existing 16-bit interval timer.

A single output instruction is used with bit patterns in M(R(X)), specifying commands to the interface. The controller, together with the interval timer,⁴ sets the timer clock rates, counts external events, and signals the CPU at preset intervals. Features of the counter/timer control interface⁴ are as follows:

- Rates from 131.3 μ s to 16.804 ms in multiples of two.
- Repeated time-out pulses at the rates given above, or a single programmed-length time-out to a maximum of 18.35 minutes.
- Choice of setting external flag and interrupt at time-out, or just external flag.

—External-event counter input available for positive- or negative-going inputs, allowing counts to 65,536, independent of CPU processing.

—Buffered time-out output that can be used to signal an external device after a preset interval.

The digital-to-analog converter interface enables CDS users to convert digital data to an analog voltage and direct it to any one of eight program-selectable output ports.

The D/A converter interface uses the 0-10 V 12-bit Datel DAC-VR12B1B D/A converter with a 2- μ s settling time. The data

format is identical to the A/D converter's. Features of the D/A interface are listed below:

—Single fixed analog output port, which additionally can be switched to any one of eight program-selectable output ports, if desired.

Microcomputer interface summary

A/D converter with 12-bit accuracy, 0-10 V range, 10-megohm input impedance, 16 program-selectable input ports, 13-microsecond conversion time, and selectable stable voltage reference.

Interrupt priority resolution logic, providing vectored interrupt servicing within 2 instruction times.

Programmable timer/event counter that can count external events and generate single or repeated time-outs at programmed rate.

Interval timer with 16-bit resolution used in conjunction with the programmable timer/event counter in a countdown mode.

D/A converter with 12-bit accuracy, 0-10 V range, 15-mA output drive current, 8 program-selectable output ports, 2-microsecond settling time, and input storage register.

Programmable power supply with 10-bit accuracy and facility to zero output and control ac connection on command.

Dot-matrix interface to standard tv, providing display in 32X32, 16X64, or 32X64 dot format.

Buffered cable to allow CDS backplane expansion into an external card nest.

Software-controlled relay matrix expandable in 32 relay increments for interfacing to "bed of nails" for PC-board testing.

Signal multiplexer to permit switching and signal conditioning of eight relay-based inputs and eight relay-based outputs.

Interprocessor interface to facilitate multimicrocomputer systems.

Program-mode I/O logic and hardware (eight switches, pushbutton, prompt LED, and audio alarm) to enable data entry during program execution.

Status and control-panel logic and hardware, consisting of nine LEDs and eight pushbuttons for displaying status information and generating a unique three-bit code.

Standard 4½-digit autoranging multimeter for measuring ac/dc current, ac/dc voltage, and ohms.

Standard digital counter for frequency, period, event and time measurement.

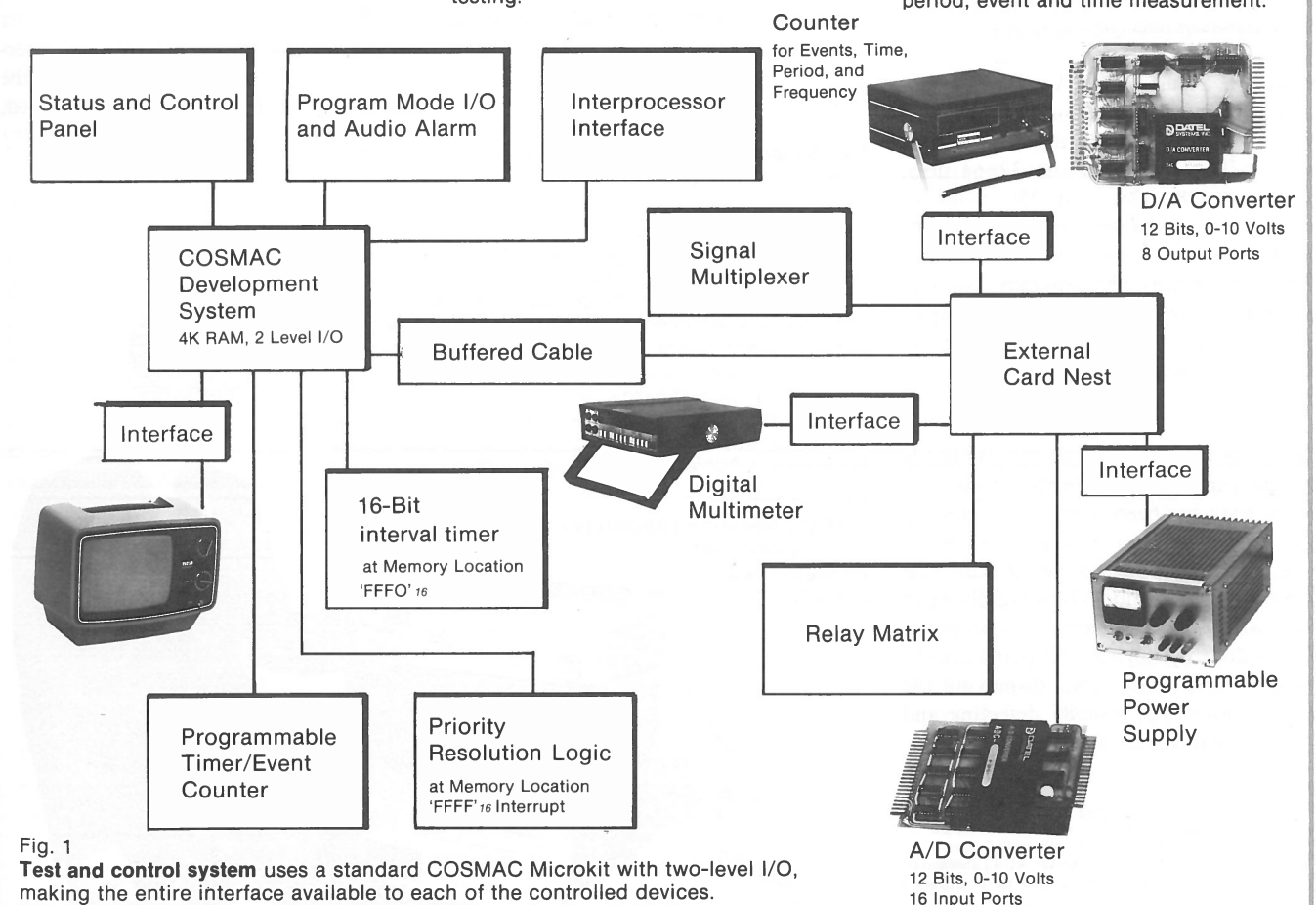


Fig. 1
Test and control system uses a standard COSMAC Microkit with two-level I/O, making the entire interface available to each of the controlled devices.

— Both hardware design and software instructions applicable to 8-, 12-, and 16-bit D/A converters.

— Input storage register that avoids output transitions due to partial field loading for 12- or 16-bit units.

Programmable power-supply interface controls a 10-bit digital-to-analog programmer.

Made by Sorenson, the programmer can, in turn, control Sorenson's complete line of programmable power supplies. A DCR 15-3B (150 V, 3 A at 300 ms) supply satisfies our high-voltage/high-current needs, but is relatively slow. To fill the need for a fast-response, lower-voltage supply, a QRD 60-1.5 (60 V, 1.5 A at 25 μ s) was chosen.^b

The interface provides, under software control, addressing logic for up to 4 DAP programmers, a *clear* signal, 10 data bits with an associated strobe, and a pair of signals used to drive the output to zero and/or disconnect the ac line from the power supply by means of an external relay. In addition, each power supply provides a flag that indicates when the output is within a 2.5% tolerance.

A tv interface provides a dot-matrix display for a standard television receiver.

Based on a custom RCA television I/O chip,^c this interface can put the display in any one of three program-selectable modes—32 $\times$ 32, 16 $\times$ 64, and 32 $\times$ 64 dots, requiring 128, 128, and 256 bytes of memory, respectively. Each dot of the display represents a bit in the CDS memory and can be modified by program. Two I/O instructions are needed, one for selecting or enabling the interface and another for setting the mode of the display and turning it on/off.

In addition to producing an extremely flexible character and graphics display, the tv interface has been very useful in testing and debugging. For example, this interface, with its dot-matrix format, can help identify faulty memory chips rapidly when used with a corresponding test program. This is done by storing a bit pattern in the memory area under test, displaying the pattern, and then visually detecting and identifying incorrect bit fields.

^aThe interface logic was designed by T.F. Lenihan, RCA Laboratories, Princeton, N.J.

^bThis chip was designed by J.A. Weisbecker, RCA Laboratories, Princeton, N.J. Additional information on the tv display is presented in Ref. 6.

A 64-wire buffered cable makes it possible to add an additional card nest to the COSMAC Development System.

Each signal goes through a logic driver at each end of the cable, thereby applying only a single load to the signal source and providing substantial drive at the signal destination. Additionally, certain lines usually connected in a "wired-or" configuration are supplied through transmission gates to provide tri-state capability.

The relay matrix can test up to 256 test points.

The relay matrix interface^d requires a minimum of two relay matrix cards to complete a circuit under test. Each card contains an array of 32 relays, any one of which can be closed under software control. The interface is most useful in applications where complete isolation is required between the measuring and measured equipment (e.g., a "bed of nails"). Each card has four bank-select switches whose settings distinguish it from a maximum of sixteen possible cards. Summarized below are the features of the relay matrix interface:

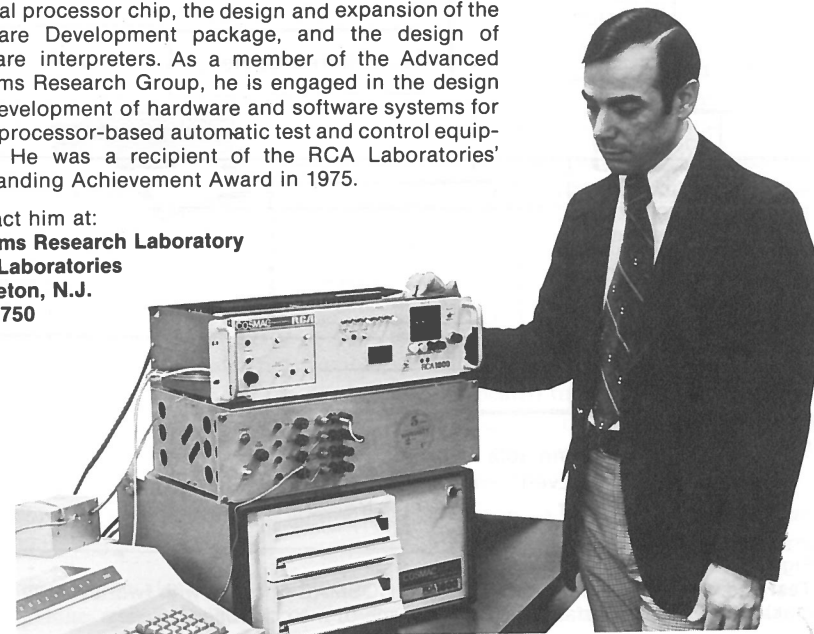
— Expandable to 16 relay cards, facilitating exhaustive testing of up to 256 test points.

^dDeveloped in cooperation with M. D. Lippman.

Paul Russo's photograph and biography appear with his other article in this issue.

Angelo Marcantonio contributed to the first microprocessor developments leading to the COSMAC and worked on the logic simulation and fault testing of the original processor chip, the design and expansion of the Software Development package, and the design of software interpreters. As a member of the Advanced Systems Research Group, he is engaged in the design and development of hardware and software systems for microprocessor-based automatic test and control equipment. He was a recipient of the RCA Laboratories' Outstanding Achievement Award in 1975.

Contact him at:
Systems Research Laboratory
RCA Laboratories
Princeton, N.J.
Ext. 2750



— Response to simple continuity/open test, directly testable under program control using external flag.

— Each relay card may be selected with or without energizing its relays.

— A single command is available to de-energize all relays, on all cards, simultaneously.

— A four-element DIP switch enables easy assignment of relay card selection numbers.

The signal multiplexer permits time-multiplexing of eight input-signal sources and eight output signals through a bank of relays.

This device is especially useful when used in conjunction with single input/single output devices such as the relay matrix. The board has space for conditioning any of the input and/or output signals required. Transmission gates are provided so that conditioned input/output signals can be connected directly to the flag lines. This may be advantageous in certain configurations, as it permits more rapid software determination of the signal state.

The inter-processor interface architecture permits interconnecting up to 256 COSMAC-based slave systems to one master.

In a master-slave configuration, all inter-processor communication goes through the master. No common memory is provided,

and each CPU will have enough RAM to handle its dedicated workload. Inter-processor communication will be via programmed-mode I/O or via DMA (for block transfers). The external flags are used for status, handshaking, and command encoding. Additional information is presented in Refs. 7 and 8.

The program-mode I/O hardware allows kit users to output a prompt signal, input a ready signal, and input 8 bits of information.

The programmable I/O hardware uses an LED, a pushbutton, and a bank of eight toggle switches in a manner quite similar to that used for the Microtutor. The input side of the CDS byte I/O card is assumed to be available for gating-in the state of the eight switches; the output side of the byte I/O card can be used to drive a two-digit LED hex display directly using the bit output lines.⁵ There is also an audio alarm available, which can be turned on/off via software.

The status and control panel is a general-purpose interface.

This interface displays eight bits of status information and reads any one of eight pushbuttons, each with a unique three-bit code. The code could be used to index a table, perform a vectored branch, select an interface, turn a device on or off, or any other function requiring a simple one-of-eight choice.

The digital-multimeter interface allows a 4½-digit DMM with a digital output option to be used as an input device to the Development System.

Interface programming for the DMM, a Fluke 8600A, is accomplished with a single output instruction. A wide variety of options is available to the user, including a mode that, independent of CPU processing, automatically updates DMM measurement data. Listed below are the relevant features of the DMM interface:

— Complete status of DMM front-panel settings available to CPU, including seventeen bits of data, three bits indicating scale, one sign bit, and one overflow bit.

— Bank of input latches allows the DMM's output to be sampled and stored every machine cycle, completely invisible to the main program with appropriate choice of options.

— Autoranging setting of DMM enables parameter being read to be anywhere

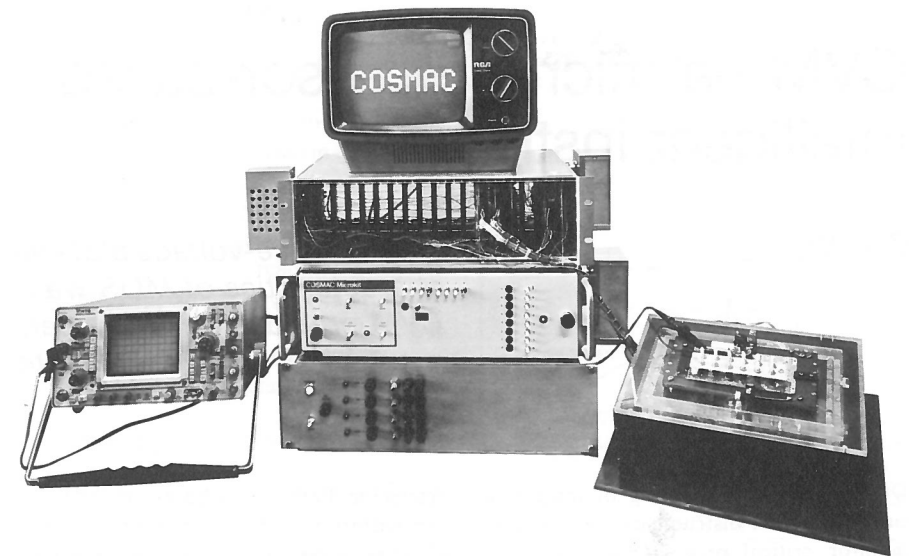


Fig. 2
Developmental hardware. Top to bottom: tv monitor, card nest, COSMAC Development System, and power supply. Scope is at left and "bed of nails" interface at right.

within the entire range of the 8600-A for all parameters except current.

The counter interface, together with a Data Precision 5470 counter, enables the CDS to make frequency, period, event, and time measurements.

This interface^e allows the CDS software to control functions of reset, run, and hold for a Data Precision 5470 counter. The specific mode of measurement is selected manually.

Ongoing work is aimed at developing additional suitable automation-oriented interfaces for the CDS.

Of particular interest is the development of an IEEE Standard Instrumentation Bus Interface for the CDS system, which would enable it to communicate with a host of standard instruments and hence permit system designers to build up various test/control systems rapidly.

Additional interfaces under development include new A/D, D/A, and printer interfaces, made possible by the rapid advances in those technologies.

Specific applications currently being investigated include closed-loop feedback-control systems and automatic testing of printed-circuit boards.

Conclusions

The basic advantages of microprocessor-based test and control systems over

^eDeveloped by C.C. Wang, RCA Laboratories, Princeton, N.J.

custom-made hard-wired fixtures is the ease with which systems can be reconfigured at the hardware level (modular) and at the function level (software changes). Since the bulk of the hardware will be identical for each test/control system, maintenance is easier, development costs are reduced, and changes are readily made. Indeed, this flexibility is the most valuable aspect of the use of microprocessors in test and control applications.

Fig. 2 presents a close-up view of the developmental hardware.

Acknowledgment

The authors gratefully acknowledge the efforts of T.F. Lenihan for his excellent construction of the prototype hardware.

References

1. *COSMAC Development System Operator's Manual*, MPM-103, RCA Solid State Division, Somerville, N.J.
2. "Adding two-level I/O interface capability to RCA COSMAC development systems," Application Note ICAN-6486, (May 1976) RCA Solid State Division, Somerville, N.J..
3. Marcantonio, A.R.; "Interrupt priority resolution circuit for use with RCA COSMAC development system," Application Note ICAN-6485, (Mar 1976) RCA Solid State Division, Somerville, N.J.
4. Marcantonio, A.R.; "Programmable interval timer and counter for use with RCA COSMAC development systems," Application Note ICAN-6482 (Mar 1976) RCA Solid State Division, Somerville, N.J..
5. Lippman, M.D.; Marcantonio, A.R.; and Russo, P.M.; "Hexadecimal display for use with RCA COSMAC development system," Application Note ICAN-6487, (Mar 1976) RCA Solid State Division, Somerville, N.J..
6. Weisbecker, J.A.; "A practical, low-cost, home/school microprocessor," *this issue*.
7. Russo, P.M.; "An interface for multi-microcomputer systems," *Proc. Fall '76 COMPCON*, Washington, D.C., Sep 1976.
8. Russo, P.M.; "Architectural trade-offs in designing microcomputer systems," *this issue*.

CVM—a microprocessor-based intelligent instrument

C.T. Wu

Capacitance-voltage plots tell a great deal about the physical characteristics of MOS wafers. The microprocessor-based CVM provides this information for on-the-spot use during manufacture and for long-term data analysis.

During the manufacturing of integrated circuits, various instruments are needed to monitor critical process parameters. In some cases, calculations must be performed on the measured quantities, and data must be logged for further statistical analysis. Since a real-time minicomputer-based system capable of handling these tasks throughout a large physical area involves a relatively large investment, an attractive alternative is to use "intelligent" instruments. Such instruments perform local calculations and then, at specified intervals, ship the formatted results to an existing main-frame computer. Microprocessors can provide intelligence to the instruments and also allow ordinary phone lines to be used as the communications link to the central computer. Simple and flexible distributed-intelligence systems can be realized this way rather effortlessly.

One such intelligent instrument is the Capacitance Voltage Monitor (CVM), based on the COSMAC microprocessor. The CVM was developed as part of an overall program in the Solid State Technology Center to achieve better understanding and control of the IC manufacturing process.

Capacitance-voltage measurements

Capacitance-vs-voltage plots are often used in the semiconductor manufacturing process to measure physical characteristics of MOS (Metal-Oxide-Semiconductor) devices.¹ For this purpose, test wafers with capacitors of known geometry are placed in a furnace together with production wafers during a run. By studying the CV plots of the resultant test samples, it is possible to

determine flatband voltages, mobile ion concentrations, bulk minority carrier lifetimes, surface recombination velocities, etc. Because the measurements are relatively simple to make, CV plots have become standard practice in RCA's MOS wafer-fabrication process for both SSTC and SSD.

CV plots show the capacitance of a sample against its dc bias. To make one, a small ac signal of fixed amplitude (millivolts) and frequency (typically 1 MHz) is usually added to a slowly-varying ramp voltage and applied to the sample. The measured reactance is then plotted as a function of the input ramp voltage. This operation can be done on CV-plotting equipments that have adjustable ramp speed and amplitude; these are available from several equipment vendors.^{2,3}

In practice, two superimposed plots are made with the same sample, one at normal temperature and one done after heating the

sample under a fixed bias. The shapes of these curves and their relative displacement contain the desired information. Quantitative interpretation of these plots involves picking off points from several precomputed charts.⁴ Specifically, in order to obtain the flatband voltage (V_{fb}), the normalized flatband capacitance (C_{fb}), and the shift after heating, three charts are needed. These results are recorded and filed if they fall within expected process limits.

Purpose of CVM

To automate this measurement-making, the microprocessor-based CVM system has been designed to monitor CV plots and compute the required results. Test results accumulated over a day are sent to a data file in TSO (Time-Sharing Option on the IBM 370 system) via an ordinary telephone line. Statistical analysis and reporting based on these data can then be performed using the full resources available in a large system.

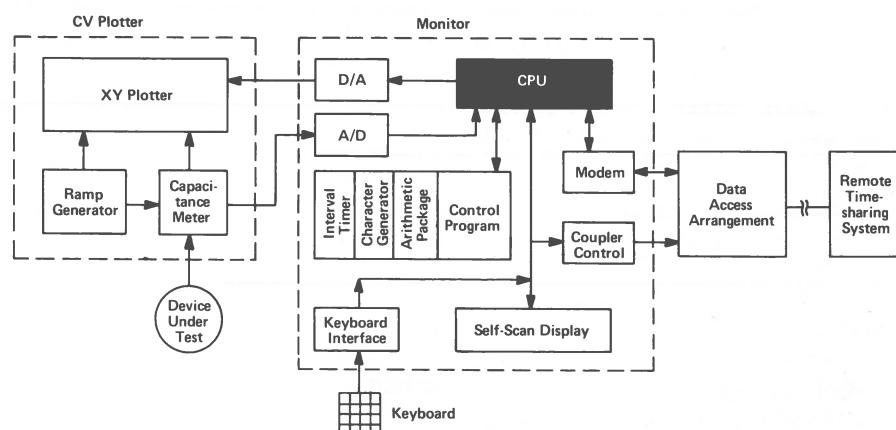


Fig. 1
Capacitance voltage monitor (CVM) system measures physical characteristics of MOS wafers. System monitors CV plots (left side of block diagram), then transmits results of a day's testing to a time-shared computer.

CVM design procedure

CVM is built up from a microcomputer nest—the COSMAC Development System (CDS)—by adding functional modules.⁵ Since the CDS has extra connectors specifically provided for user-developed boards, systems directed toward particular applications can be constructed easily. To do this, each board is breadboarded and then debugged in CDS with the aid of an interactive debugger, provided with CDS. By ensuring that each hardware component operates properly in real time, final debugging becomes simpler.

Applications software for the system was developed in parallel with the aid of the COSMAC System Development Program (CSDP),⁶ and most of the functional flow and computations were simulated with CSDP. Routines dealing with peripheral control and real-time responses, however, were handled with the debugger in CDS.

System components

CVM consists of two parts, a Boonton CV Plotter and a COSMAC-based monitor. Fig. 1 shows the monitor's functional components; each block in the diagram represents a single PC board (except the one marked "CPU," which is the CDS microcomputer itself).

The following I/O modules, when added to the CDS, form the CVM:

- A/D converter—used to sample a pair of voltages corresponding to the x-y plot
- D/A converter—used to drive the x-y plotter
- Self-scan display—used for displaying computed results
- Keyboard interface—used for operator entry of parameters
- Interval timer—used for updating time-of-day clock
- Data coupler control—used for data access and dial-up of computer system
- Modem—used to modulate and demodulate signals to the telephone line

In addition to the I/O modules, two ROMs are needed for computational purposes. Each resides in a PC card; they are:

- Arithmetic Subroutine ROM—16-bit arithmetic subroutines for multiply, divide, and conversions
- Character Generator ROM—5×7 dot-matrix representation of ASCII-coded characters

Control signals for most of these I/O modules are derived from the CDS I/O decoder. Here, eight output ports and eight input ports can be supported in a one-level I/O structure. Apart from these, I/O can also be mapped as memory data, as in the interval timer. In this module, controls are derived from memory read/write signals and "time" appears as two bytes of RAM. Depending on the particular I/O device, one of these methods may prove more convenient for the programmer.

System operation

Fig. 2 is a simplified flowchart of the CVM system. As indicated, the two main tasks that the monitor performs are local computation of CV measurements and data-logging to a remote computer. To start, an operator initializes the CVM at the beginning of the day by data and time entries. Then the CVM makes a series of measurements until the end of the day, when it transmits the collected data automatically.

Operation of the system is as follows. For each test, CVM asks the operator to enter two identification numbers associated with it. First, the display shows:

SAMPLE #=

After entering the number through the keyboard, the display shows

TCC #=

A second number can then be entered. The display will now show:

ENTER COMMAND

There are four options in responding to this message. To get a continuous display of time (updated every second), the user

Chin Tao Wu has worked on various aspects of computer systems design since joining RCA in 1965. He has been interested in microprocessors since 1971 and was active in the earlier support work for RCA's COSMAC microprocessor. Mr. Wu has received three RCA Laboratories Outstanding Achievement Awards, has published a number of technical papers, and holds several U.S. patents.

Contact him at:
LSI Systems Design
Solid State Technology Center
Somerville, N.J.
Ext. 6061

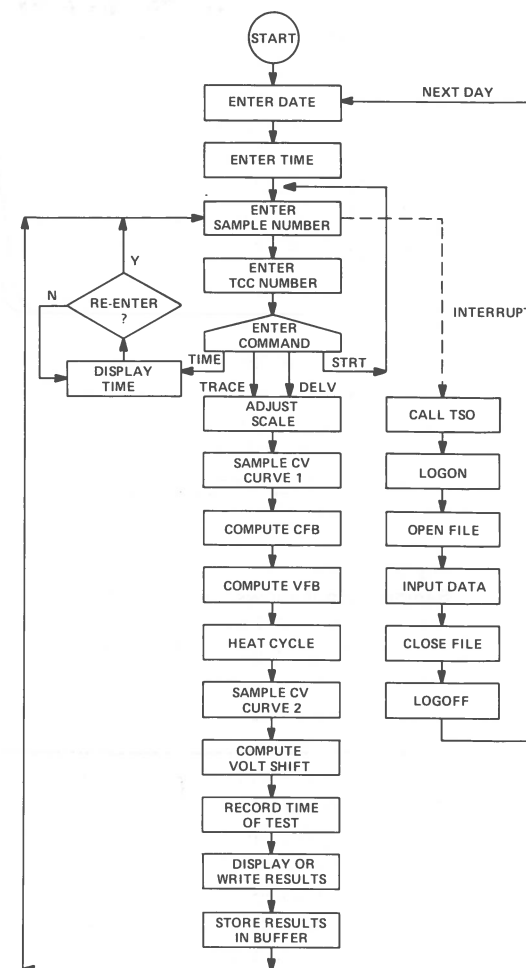
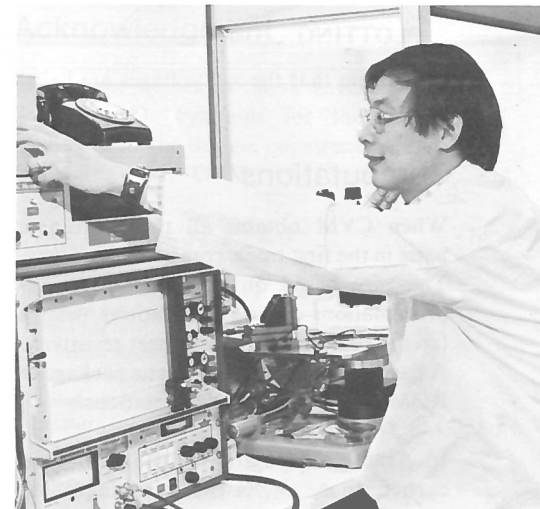


Fig. 2
CVM takes two curves for devices, one at normal temperature and one after heating, then compares them. Data is logged out to time-shared computers (TSO) for analysis upon interrupt at end of a day's testing.



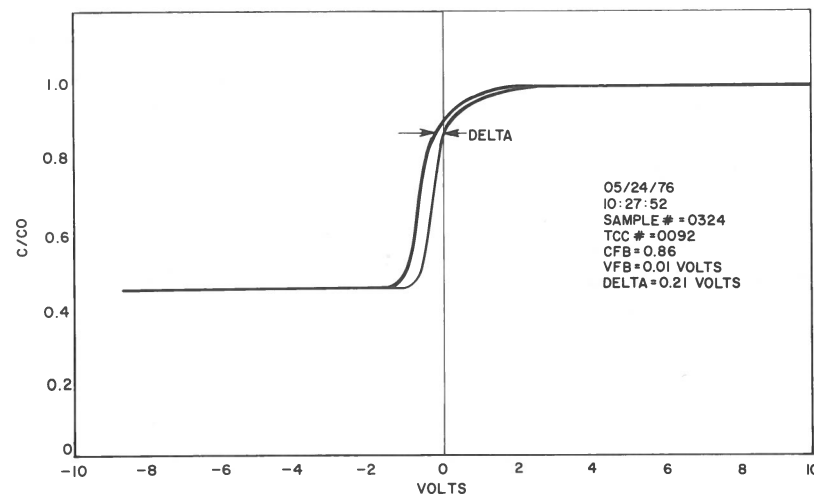


Fig. 3 Sample output shows three directly-computed data points. CFB is the normalized flatband capacitance, VFB is the flatband voltage, and DELTA is the voltage shift after heating. Old system required picking off values from precomputed charts.

depresses the key marked 'TIME'; to perform a measurement and have the results shown on display, he hits the key marked DELV; to perform a measurement and have the results written with the plot, he hits the key marked TRACE; and, to enter a new set of test identifications because of entry error, he hits the key marked STRT.

During an actual measurement, the scales in the x-y plotter and the capacitance meter usually have to be adjusted in order to obtain a plot with proper y-deflections. To allow for this, CVM shows:

ADJ SCALE & STRT

After adjusting the scales, the operator then hits STRT to begin analog sampling of the x-y plot by the monitor. During this process, CVM shows:

PLOTTING

to indicate that the x-y voltages are being sampled.

Computations

When CVM obtains all the coordinate pairs in the first trace, computations begin. They are based on piecewise linear approximations of the CV equations, using a fixed oxide thickness and sheet resistivity. A fixed-point, 16-bit arithmetic package in ROM is used for these calculations.

In the computation, the flatband capacitance, C_{fb} , is first computed as a

function of the minimum measured capacitance, C_{min} . Then, from C_{fb} , the corresponding voltage from the CV trace is picked off as V_{fb} . Both C_{fb} and V_{fb} are obtained from the first trace. After this, the sample is heated under a fixed bias and then cooled.

Sampling of the second CV trace begins after the heat cycle. Here, the voltage corresponding to the C_{fb} of the previous trace is found from the curve. This result is subtracted from the previous V_{fb} to obtain the voltage shift Δ between the two traces. C_{fb} , V_{fb} and Δ are then stored in memory together with test identification and time of test for later logging.

Fig. 3 is a sample of a CV trace with the accompanying data calculated and plotted by the CVM.

Character plotting

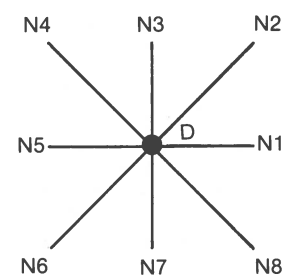
The way in which CVM writes alphanumeric characters on the x-y plotter illustrates an interesting capability of a microprocessor-based system. It is an example of performing a required function by combining relatively complex software with simple-minded I/O hardware. Fig. 4 shows the flowchart of the character-plotting routine.

The plotting operation requires a D/A converter to generate a pair of deflection voltages. Fig. 5 shows the functional block

diagram for the converter and its interface to CDS.

From an OUTPUT instruction, the 8-bit data is split into two 4-bit fields in the converter interface. One of these 4-bit fields is decoded to generate commands for multiplexing control and pen up/down motions; the other 4-bit field contains the data for the D/A converter. Two analog "sample-and-hold" gates hold the x- and y-voltages between conversions.

Pen plotting movement consists of tours through the connecting dots forming the dot-matrix of an alphanumeric character. Travel preference is arbitrarily assigned to the first-encountered dot in a counter-clockwise sweep starting from the x-axis.



For the example above, if a dot D has two neighbors, N2 and N3, the pen movement will be from D to N2.

To start a tour, the pen begins to scan from the lower left corner of a 5x7-dot character image. As the dot-matrix image is traversed, dots in the path are erased one by one so that the remaining dots are unexplored parts of a character. At the termination of a linear segment, the last dot will have no immediate neighbors. Scanning then begins again from the lower left corner; this procedure is repeated until all dots have vanished.

To obtain various character sizes, scaling can be done by changing the constants assigned to the x- and y-deflections.

Data logging

An interrupt occurring at a fixed time daily starts data logging. First, the TSO system phone number is dialed automatically and DVM logs itself on as an interactive user. Then, a data file is opened and the collected data is entered. Afterwards, CVM closes the file and logs off the system. These actions are performed in an interval of several minutes by software that mimics a normal TSO user.

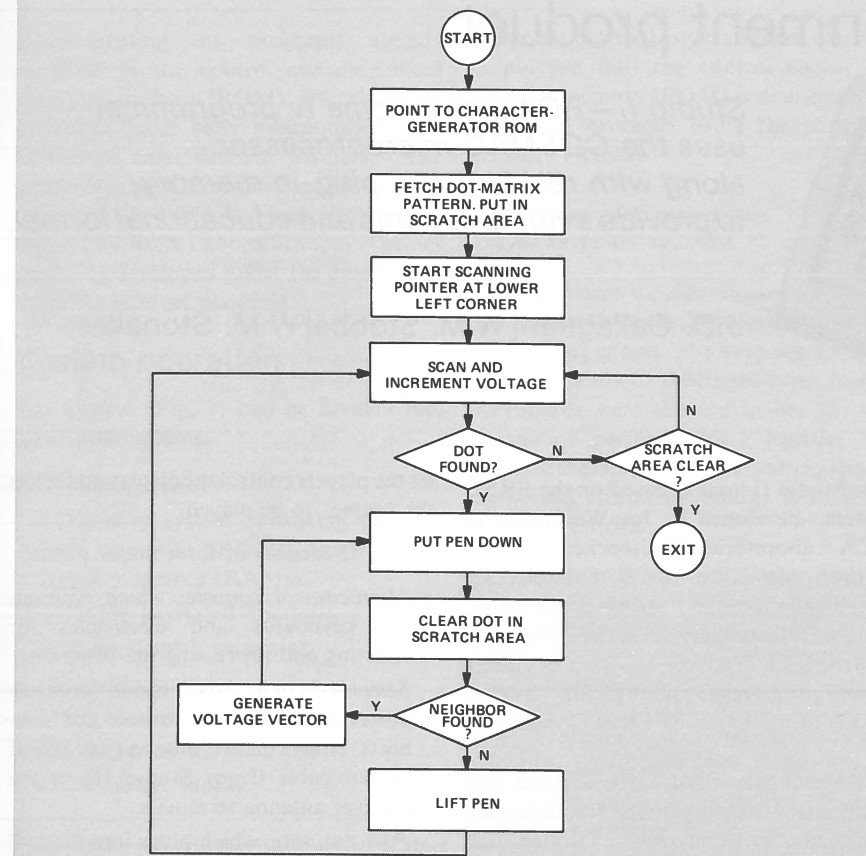


Fig. 4 Connect-the-dot character plotting is done with simple hardware and relatively complex software. Pen scans 5x7 character image to find the linear segments.

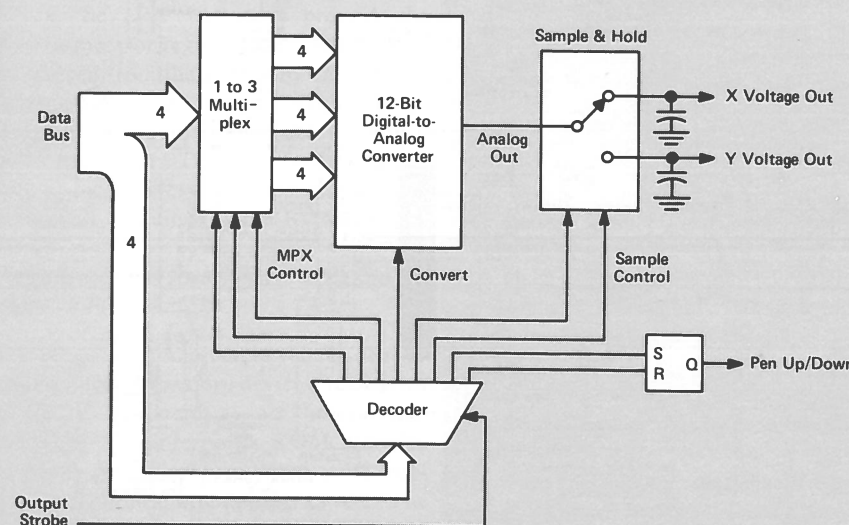


Fig. 5 Plotting system generates x-y deflection voltages through D/A converter.

Communications with TSO are carried out in asynchronous ASCII code. Since ASCII code cannot accommodate eight data bits per character, binary data are coded so that each character carries only four bits of information. Therefore, hexadecimal numbers are simply coded as the corresponding ASCII characters 0, 1, . . . E, F. Conversion to recover the four bits is relatively straightforward in this coding scheme.

The logged data constitutes a basis from which statistical analysis and quality control can be realized. For this, large computers are usually better than microcomputers because of the need for crunching large quantities of numbers. Moreover, the file-storage capability of a large main-frame computer is necessary for keeping a large data archive.

Since process degradations must be detected and corrected as soon as possible, process data collected on the factory floor are crucial. For a complex process such as manufacturing integrated circuits, continuous distributed monitoring of various parameters is necessary for fast corrective actions.

Conclusions

The CVM system shows that microprocessors can provide intelligence to process-monitoring and control instruments. This intelligence frees the operators from manual tasks and also allows the collected data to be communicated to a remote central computer, where it can be processed and filed. Simple and flexible distributed intelligence systems can be realized with relative ease for the control of complex processes.

Acknowledgment

The author wishes to thank K.H. Lee, J. Kau and D. Fraipont for their contributions in the design, construction and debugging of the CVM.

References

1. Zaininger, K.H.; and Heiman, F.P.; "The CV technique as an analytical tool," *Solid State Technology*, Vol. 13, No. 5-6, (May - Jun 1970).
2. CV Plotter, Model 179A, Boonton Electronics Corp.
3. CV Plotter Model 410, Princeton Applied Research Corp.
4. Zaininger, K.H.; and Heiman, F.P.; *op. cit.*
5. COSMAC Microkit Operators Manual, MPM-203, RCA Solid State Div., Somerville, N.J.
6. Program Development Guide for the COSMAC Microprocessor, MPM-202, RCA Solid State Div., Somerville, N.J.

Studio II—using COSMAC in a home entertainment product



Studio II—RCA's new home tv programmer—uses the COSMAC microprocessor, along with resident and plug-in memory, to provide a versatile game and educational format.

J.D. Callaghan|W.M. Stobbe|W.M. Stonaker

RCA recently entered the consumer video game market with a tv programmer called "Studio II," which is currently being marketed on a selected basis, with national distribution scheduled for mid-year.

RCA's new single-chip version of the COSMAC microprocessor, the CDP1802, provides central computer control for the various game, educational, and other entertainment programs offered with Studio II. Thus, a very high level of program "sophistication" can be achieved, limited only by the solid-state memory capacity. The product remains flexible and updatable through the simple expedient of plug-in program cards.

The Studio II logic is based on the FRED system—developed by Joe Weisbecker of RCA Laboratories. Weisbecker and P.K. Baltzer, also from RCA Laboratories, developed the initial software programs for Studio II. In January 1976, the Distributor and Special Products Division engineers were given responsibility for product design of the modified FRED system.

The result was Studio II—a game that connects to the vhf antenna terminals of a standard television receiver. Two ten-digit keyboards, located on the main housing,

let the players control the display and select the "game" to be played.

Studio II consists of three major pieces:

Main-control console, which contains the keyboards and electronics for selecting and processing the programs.
Selector-switch box, which controls power to the control console and connects either the tv antenna or the rf output cable (from Studio II) to the receiver antenna terminals.

Power supply, which plugs into the wall ac outlet and supplies 9 Vdc to the

control console via the selector-switch box and rf output cable.

Supplementing the programs already available in the control console's fixed read-only memory (ROM), several plug-in cartridges have been programmed with additional entertainment formats. The games that have been developed to date are described in Table I. These games were devised by RCA Laboratories; new games are being developed under the direction of D&SPD product planning.

System operation

The system (Fig. 1) can be divided into eight subfunctions:

- 1) Microprocessor,
- 2) Memory system consisting of read-only memories (ROMs) and random-access memories (RAMs),
- 3) Player/keyboard interface,
- 4) Video display generator,
- 5) Sound circuit,
- 6) RF oscillator/modulator section,
- 7) Selector-switch box, and
- 8) DC power supply.

The heart of Studio II is RCA's new COSMAC microprocessor, the CDP1802.

The CDP1802 is an LSI CMOS, 8-bit, register-oriented CPU. For this application, it is operated at 5 Vdc and at a clock frequency of 1.760 MHz.

When the *clear* button is pressed, the microprocessor registers are reset. A game is selected from the ROMs by entering the appropriate codes via the keyboard, as outlined in the instruction sheet supplied with Studio II. The game starts as the microprocessor executes the program instructions contained in the ROMs.

The memory has 2048 bytes of ROM and 512 bytes of RAM.

For programs contained in the resident or external ROM memory devices, 1024 bytes of ROM are used as an "interpreter" language. The interpreter ROM provides common game display patterns, scorekeeping, subroutines, alphanumeric, etc. The remaining 1024 bytes of ROM are mask-programmed resident game formats.

The RAMs are used for tv refresh, stack, and variable storage. Data bytes can be written into the RAM (memory write, MWR) or read out of the RAM (memory

read, MRD) on command of the microprocessor.

When an external entertainment cartridge is plugged into the control socket, the resident-memory (ROM) is deselected and operation proceeds using the cartridge-memory (ROM).

Two 10-key pads provide the player/keyboard interface.

The keyboards were developed specifically for Studio II by the Deptford mechanical engineering group. The keys are arranged in the standard telephone-type format. Keyboards were selected in lieu of other means of player-interface because they provide the flexibility of permitting specific numerical entries as well as motion and direction entries.

The decision to make, rather than buy, the keyboard was motivated by several design objectives:

- 1) Proper button size and spacing for this type of product,
- 2) Ease of installation and assembly,
- 3) Good reliability, with a life test in excess of a million cycles per button,
- 4) All the above at a low cost commensurate with total product cost.

Also, tests showed that available low-cost keyboards could not withstand the pressures developed during the excitement of game-playing.

The keyboards are bused to a latched 4-bit-to-16-line decoder integrated circuit which is under program control for switch

Table I

Five resident game formats and four additional plug-in cartridge programs have been developed. Additional programs are currently being written and new ones will continually be developed.

The five resident programs included in the instrument are:

Doodle transforms the tv into an electronic blackboard.

Patterns can be programmed to design literally millions of interesting and attractive patterns on the tv screen.

Computer bowling starts with a bowling alley on the tv screen. The bowling ball moves up and down, until the player presses a key that sends the ball down the alley to strike the bowling pins. Pressing different keys will hook the ball in either direction or send it straight.

Freeway puts two racing cars on the screen. One racing car is controlled by the computer. The player controls the other car, and steers by keyboard control, to avoid collisions with the computer-controlled car. Accidents slow the player's car down and lower the score. This is a race against the clock, in a test of reaction time and coordination.

Addition allows one or two players to use either or both keyboards. A three-digit number appears on the screen. Each player must quickly add the three numbers, then enter the total on the keyboard. The faster the entry, the higher the score.

Plug-in home games presently completed are:

Space War—This module provides two games:

Horizontal intercept is for one player at a time who has available twenty steerable rockets to shoot at a fleet of fifty enemy spaceships. Hitting the small, fast-moving targets gives a better score.

Vertical intercept allows two players to attempt hitting a vertically moving target by adjusting the trajectory of their missiles.

Fun With Numbers—This module provides three games:

Guess the number puts a single player against the computer, which selects a random three-digit number. As the player guesses what the number might be, the computer gives clues as to how close the guess is. If the player can't guess the numbers within twenty tries, the computer-generated number is displayed and the player loses.

Guess the number allows each player to load a secret number into the computer for the other player to guess. Clues are given as in the one-player game by the computer, and the first player to guess the other's number wins. A good test of logical thinking.

Reverse displays the digits 1 through 9 in random order. The object is to unscramble the numbers in as few "moves" as possible. A "move" consists of reversing groups of numbers according to the key number pressed. The game allows thirty "moves" before it declares the player a loser.

TV School House I (yellow series/orange series)—Each series contains both Social Studies and Mathematics sections. The *yellow* series is elementary in skill level while the *orange* series is more advanced. Comprehensive handbooks are included. This series was developed with the guidance and help of education specialists.

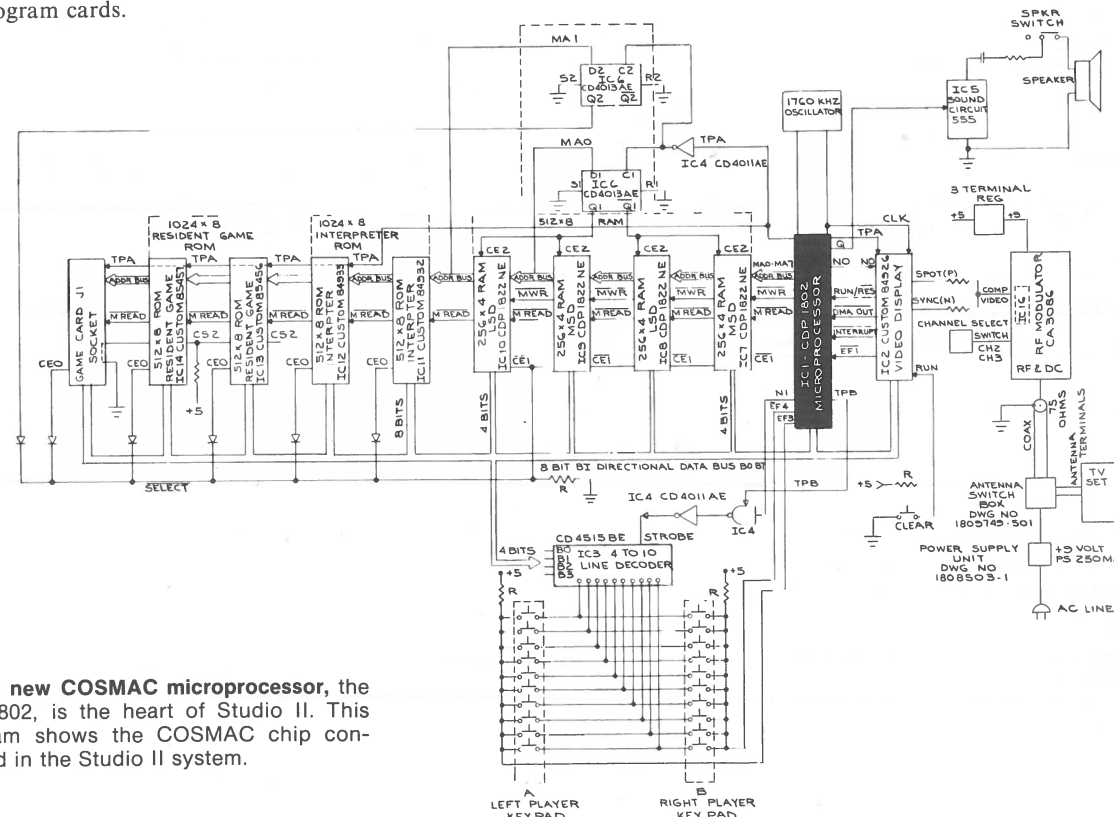


Fig. 1
RCA's new COSMAC microprocessor, the CDP1802, is the heart of Studio II. This diagram shows the COSMAC chip connected in the Studio II system.

sampling. A second microprocessor output instruction is used to sample any switch by connecting it to the "flag" lines (EF3 or EF4). Programs are written so as to sample any combination of keyboard A or keyboard B switches in any order.

The video display generator is a custom IC, CDP1861.

The video display generator, RCA CDP-1861, is a customized integrated circuit designed to work with the CDP1802 microprocessor. Direct Memory Access (DMA) and Interrupt signals from the display generator to the microprocessor provide a continuous 32x64-dot display matrix.

A transistor-array IC, CA3086, is used as the oscillator/modulator.

Two separate oscillators are used to generate the rf carriers for operation on either channel 2 or 3, to avoid possible interference from local broadcast stations.

The entire assembly is very carefully laid out and shielded to meet the very stringent radiation requirement of 15 $\mu\text{V}/\text{m}$ at 1 meter. This assures that no interference will be generated on nearby tv sets.

The modulation percentage is limited to 55% so that the tv receiver will not see a saturated white level which might cause picture tube burn-in from lack of picture motion during prolonged use.

The selector-switch box is the interface between the tv antenna, tv receiver, the Studio II rf-output cable, and the dc power supply.

The selector-switch box normally mounts on the rear of the tv receiver, close to the antenna terminals. In the "Studio II" position, the switch disconnects the tv antenna from the receiver, connects the rf output into the receiver, applies power to the Studio II via the rf cable, and transforms the unbalanced 75-ohm Studio II output to 300 ohms, balanced for the tv receiver.

This function must be done and still meet the FCC rules, which state that the rf transfer switch must maintain 60-dB isolation between the Class I tv device and the antenna.

The power supply is a UL approved wall plug-in type providing 9Vdc at 250 mA.

The dc output cable plugs into the selector switch box where it is switched into the

main Studio II console by the selector switch. The 9Vdc is converted to the 5Vdc needed for the circuitry in Studio II by means of a 3-terminal voltage regulator on the main pc board.

"Beep" sounds add another sensation to the game action.

A 555 timer-oscillator circuit, activated by a microprocessor-controlled signal, generates "beep" sounds at appropriate times during game action. The beeps are not intended to represent any particular sound. A switch is provided so that the sound can be shut off if desired.

List price

Optional retail price of Studio II is \$149.95. The *Space War* and *Fun-With-Numbers*

cartridges are optionally priced at \$14.95 each. *TV School House* will have an optional list price of \$19.95, since a comprehensive program manual is included.

Conclusion

In Studio II, RCA is offering a low-cost home video game that can be programmed for an almost unlimited variety of games by inserting new ROM cartridges; new game cartridges are being developed on a continuing basis. This capability should offer the consumer a clear advantage over the subtle variations on the "ball going back and forth" type of game that has been available for the past year or so. The range of Studio II games is limited only by the two dimensions of the tv screen, the solid-state memory capacity, and the imagination of the game-programmers.

Final manuscript received February 23, 1977.

Bill Stonaker, as a Principal Member of D&SPD's Engineering Staff, is responsible for digital circuit designs, including the Studio II logic design and circuit configuration. He joined D&SPD in 1975, after having been associated with RCA's Equipment Design and Development Engineering Group in Harrison.

Walt Stobbe, as a Principal Member of D&SPD's Engineering Staff, is responsible for various rf and digital projects including the Studio II rf section. Prior to joining D&SPD in 1975, he was responsible for design engineering and development of solid-state test equipment products at RCA in Harrison.

Dave Callaghan, as Manager of Engineering, is responsible for all product development at D&SPD; these include tv accessories, color tv test jigs, CB radio, and automotive audio products. The Studio II home tv programmer is the most recent product.

Authors **Stobbe**, **Callaghan**, and **Stonaker**. Contact them at:

Engineering, Distributor and Special Products Div., Deptford, N.J. Ext. PT-531



A microcomputer-controlled spark advance system

A. D. Robbi | J. W. Tuska

A microcomputer can calculate and control the exact time of spark firing as a function of speed and load to give better engine timing accuracy and dynamic response than a mechanical advance-retard system. Microcomputer control also allows the engineer to build in additional features, such as automatic misfire protection, and has noticeably improved "driveability" in road tests.

DURING the 1950's solid-state devices capable of switching the primary current of an automobile ignition coil became available. Ignition systems using these transistors, normally with a specially designed ignition coil, were built by enthusiasts and offered on the after-market in either kit or finished form by many suppliers. These systems retained the conventional mechanical breaker points, but the breaker points switched transistor base current rather than coil current. Well-designed "transistor ignition systems" increased the amount of energy delivered to the spark plugs and extended the mileage that the vehicle could be operated without significant performance deterioration.

Breakerless electronic ignition systems became available as a factory option during the 1960's, and by late 1973 a significant percentage of the new automobiles produced in North America were assembled with breakerless solid-state ignition systems as standard equipment. It was at this time that we decided to investigate and implement a fully electronic spark timing system. The authors, and their colleagues, are not automotive engineers; and we do not profess to know all the answers to the

problem of controlling a modern internal combustion engine so that it provides excellent driveability and performance, certifiably low emissions, and improved fuel economy. We believed that a system in which the exact times of spark firings were calculated and controlled electronically, as a function of engine speed and load, would offer improved timing accuracy and faster response than conventional advance-retard systems with centrifugal weights and mechanically moved sensors in the distributor.

We further believed:

- 1) that a digital system based on a microprocessor would give us the capability to provide the required spark timing and the flexibility to easily modify the spark timing based on additional sensor inputs or vehicle information;
- 2) that a microprocessor-controlled spark advance system would allow the automotive engineer new freedom to employ sophisticated control algorithms in a cost-effective way, via software, for ideal ignition timing over a greater range of conditions, both environmental and operational, than is possible with mechanically constrained systems; and,
- 3) that a true test of the suitability of a CMOS microprocessor, the RCA COSMAC, for engine control, was to build such a system

and operate it in the hostile environment of an automobile.

The system was designed to match the ignition specifications for the 302-cubic-inch displacement, 8-cylinder engine of our instrumented station wagon, and installed on that vehicle. A year of vehicle operation over a variety of driving conditions, specific tests, and demonstrations has confirmed our beliefs.

In the subsequent sections we describe the spark timing problem and its mechanical solution, the approach we have taken, and the performance of our system.

Spark timing

Spark timing is a critical parameter for the modern internal-combustion gasoline-fueled engine because the spark initiates the combustion process. The combustion of the gas-air mixture in the cylinder of an engine should cause the maximum pressure to occur very near the time when the piston has completed the compression stroke and is about to begin the expansion stroke. The relative timing of the spark must be variable because the speed of combustion of the mixture is not the same under all engine operating conditions, and engine operating speeds vary by a factor of approximately 10 from idle to maximum speed. Thus, at a 500-rpm idle speed, 3° of relative spark timing corresponds to 1 millisecond of time; while at 5000 rpm, 1 millisecond of time is equivalent to 30° of relative spark timing.

Factors influencing the combustion process include the air/fuel ratio, the engine load, the throttle opening; and on recent engines, the amount of exhaust gas recirculation (EGR) added.

Spark timing is normally referred to as the angle in degrees of crankshaft rotation between the angular position of the crankshaft when spark occurs and its position when the piston passes from the compression stroke to the power stroke (top dead center, or TDC). Spark occurring on the compression stroke is referred to as "advanced" as it occurs before top dead center (BTDC); while a "retarded" spark occurs on the power stroke and is after top dead center (ATDC). Fig. 1 illustrates these relative piston-crankshaft positions.

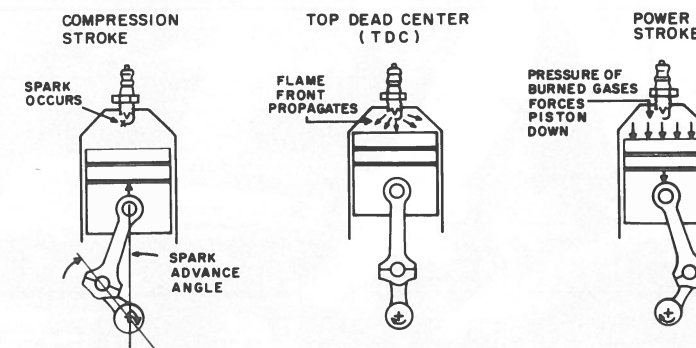


Fig. 1 — Crankshaft vs. piston positions. Spark shown for advanced spark timing (before top dead center).

Nearly all automotive ignition systems, whether conventional breaker-point/coil or variable-reluctance-triggered solid-state types, change the spark advance angle by mechanical means. Changes that are proportional to engine speed (or depart from that relationship as needed) are usually obtained by a centrifugal device that rotates the reluctor (or breaker-point cam) with respect to the distributor shaft in the direction of shaft rotation. This relative rotation increases with speed and generates a trigger (or opens the points) sooner than otherwise, resulting in an earlier (advanced) spark firing.

Changes to the spark advance angle to adjust for torque load or throttle angle are generally accomplished by sensing one or more pressures, (e.g., the intake manifold or carburetor venturi pressure). The differential between the sensed pressures, or sensed pressure and atmospheric pressure, is applied via a flexible diaphragm and rod to control the partial rotation of the nominally stationary plate in the distributor that supports the trigger coil (or breaker points). Rotating this plate opposite to the shaft rotation increases the spark advance angle, and since the sensed pressure is less than atmospheric for normal engines, this component of spark timing is frequently referred to as "vacuum spark advance."

The "vacuum spark advance" and "centrifugal" (or rpm) advance components are combined in the distributor to produce the desired spark timing. Since these components are mechanically generated, their operation is subject to wear, friction, and fatigue of the restoring springs.

Thus far we have defined advanced spark, retarded spark, TDC, and have identified the rpm and vacuum components of specific spark timing. These and two additional terms—timed maximum advance (TMA), and REF, the minimum spark advance—are shown in Fig. 2. REF is the position of spark firing when no rpm or vacuum advance components of spark timing are considered; an engine is usually timed for this spark to occur at the designated crankshaft position by rotating the distributor housing relative to the engine block. TMA represents the most advanced time of spark firing and is the sum of the maximum values of rpm and vacuum advance.

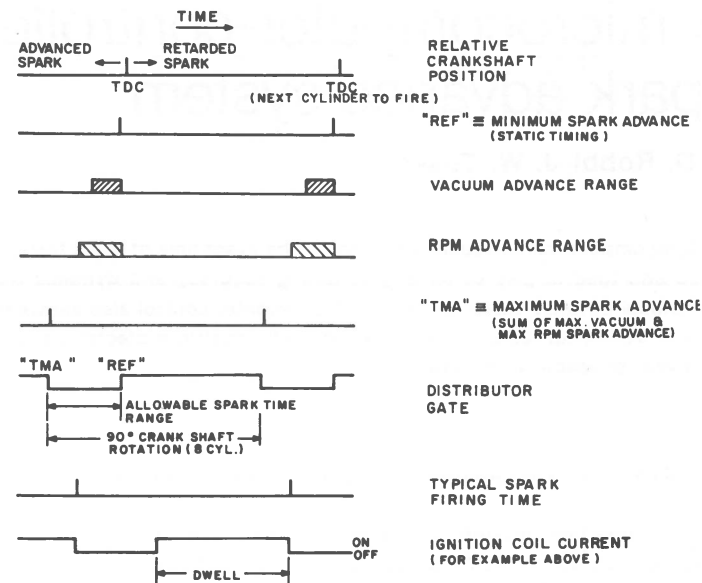


Fig. 2 — Spark timing.

James W. Tuska, LSI Systems Design, RCA Laboratories, Princeton, N. J., received the BS in Electrical Engineering from the University of Pennsylvania in 1956. In 1959 he joined RCA Laboratories, where he has been engaged in memory switching circuit research, permalloy magnetic sheet and plated magnetic wire memory developments, the integration of MOS devices for use with monolithic ferrite memories, and the development of fast, high-voltage switching circuits for computer terminal alphanumeric color displays. In 1967 he was awarded an RCA Laboratories Achievement Award for his work on high-speed laminated-ferrite memory systems. More recently, his research involved the application of the MNOS memory transistor as the storage element in an integrated non-volatile electrically alterable random-access read-only memory. He is presently engaged in the application of integrated circuit technology and digital information processing to automotive electronics systems. Mr. Tuska holds two U.S. patents and has co-authored several papers.

Anthony D. Robbi, Sr. Member, Technical Staff, LSI Systems Design, SSTC, Somerville, received the Ph.D. from Carnegie Institute of Technology in 1961. He has also received much on-the-job education in computers since joining RCA in 1961. Since 1971 Dr. Robbi has worked in microprocessor research, and has participated in CEE-55, a video-tape training course on microprocessors. During his career with RCA he has received three RCA Laboratories Outstanding Achievement Awards, has published many technical papers, and has submitted numerous patent disclosures. Dr. Robbi is active in IEEE, especially in the social implications of working as an engineer. Away from work he serves as a school board member and enjoys the outdoors.

Final manuscript received January 9, 1976.

Jim Tuska (left) and Toni Robbi check the timing on their 'electronic car.'

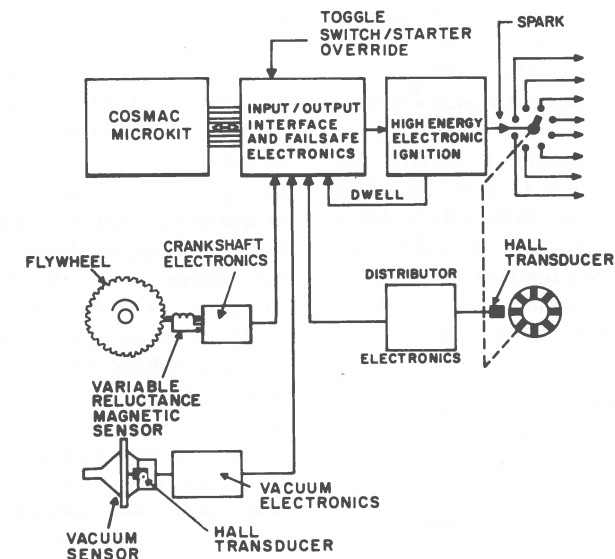


Fig. 3 — COSMAC-controlled ignition electronics.

Up to now we have defined our spark timing in terms of crankshaft degrees. The term *dwell* is generally referred to in ignition system discussions as an angular measure of the time during which current flows in the primary winding of the ignition coil. This term, however, is measured in degrees of distributor shaft rotation, which is half the crankshaft speed in a 4-cycle engine. The formula below indicates that for a typical 8-cylinder dwell of 30°, current flows in the ignition coil primary for 2/3 of the available time.

$$\text{Duty cycle} = \frac{\text{indicated dwell}}{\text{available time per cylinder}}$$

$$= \frac{\text{indicated dwell}}{360^\circ \div \text{no. of cylinders}}$$

System overview

The hardware constituents of the experimental spark control system are shown in Fig. 3. The digital system output is a trigger signal to a commercial high-energy electronic ignition.² The ignition electronics determine the spark duration and initiate dwell, and this information is fed back to the digital system. Following the manufacturer's specifications,³ the spark advance is computed as the sum of two components, vacuum advance and rpm advance. Vacuum is determined by measuring a Hall-effect voltage, which is a function of diaphragm position. Rpm is determined by measuring the duration of

a synchronizing signal derived from the distributor. This signal also serves as a reference point for counting degrees to the desired firing angle and as a timer for default firing.

The flywheel, with 164 teeth on its circumference, is an excellent source of crankshaft angular rotation and position, as it is rigidly attached to the crankshaft. The crankshaft electronics converts the tooth count to a 328-cycle-per-crank-revolution wave. Each pulse may be considered to correspond to a "tooth degree", which is thus 360/328 actual crank degrees. To fire an advanced spark the computational system performs two primary tasks: computing the advance (rpm + vacuum) in tooth degrees, and counting tooth degrees from a reference point, TMA, to the desired firing point. The computation is rapid enough to allow a fresh computation for each spark. This, and the fact that spark is synchronized with the crankshaft, which may accelerate as much as 20% per spark firing (1/4 revolution), gives rise to an excellent dynamic response.

The possibilities of misfiring (too early or too late) or an omitted firing are reduced by a failsafe strategy employed in the interface. Computer-initiated sparks are gated by the TMA-REF signal, thus preventing misfiring. If a computer-initiated spark does not occur in the interval, the interface initiates spark at REF. This mode is intentionally used when the starter is active or if an override

toggle switch located in the passenger compartment is closed.

A flow chart for the main program loop accomplishing the spark computation and control is shown in Fig. 4.

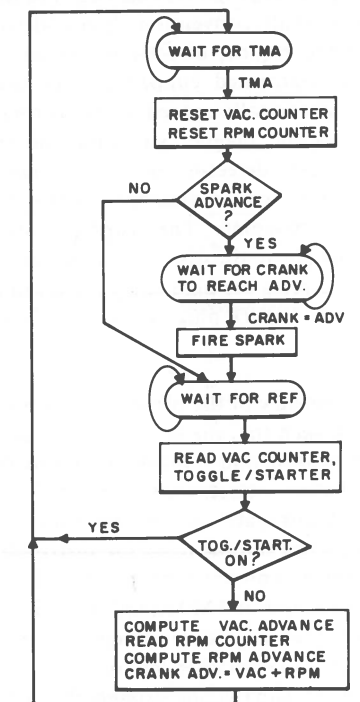


Fig. 4 — Ignition program flow.

Peripheral circuits and sensors

Two of the peripheral sensors shown in Fig. 3, the distributor and the vacuum sensor, are experimental devices fabricated by members of the Laboratories Automotive Systems Group based in Somerville.* While more conventional sensors, representative of the state of the art of the most recent automotive production sensors, could have been used in our system, these experimental types were chosen as part of an ongoing program to investigate and develop effective solid-state sensors fitting the anticipated needs of the automotive industry.

The distributor for this experimental system is greatly simplified when compared to the distributor of a conventional ignition system. The experimental dis-

*J. Olmstead, P. Del Priore, and L. Herrod

tributor is illustrated in Fig. 5. There is no movable baseplate for vacuum advance changes, and no centrifugal advance mechanism with weights, pivots, and springs. A standard distributor housing was used to facilitate installation and to accommodate a cap and rotor designed for high-energy ignition service. The distributor shaft, conventionally gear-driven from the engine, directly drives an inverted segmented cupped ferrous disc. The eight equally-spaced segments rotate between the magnet structure and the Hall-effect detector of the variable-reluctance sensor attached to the distributor baseplate. The distributor shaft extends above the disc to drive the rotor, which directs the high-energy ignition to the proper spark plug as in conventional systems.

The geometry of the rotating segments is chosen such that the sensor output, after amplification and standardization by the distributor electronics, yields the "distributor gate" waveform shown on Fig. 2. The output from the distributor electronics is digital, with state transitions of high to low at TMA, and low to high at REF. The amplitude ("high" = +5V, "low" = 0V) is independent of speed. This output is capable of driving the high-energy electronic ignition module directly without the input/output interface or failsafe electronics. A firewall toggle switch is included in our installation should the need for the direct connection arise, and as a convenience for moving the vehicle when the COSMAC-controlled system is removed for bench testing or modifications. (Spark firing occurs at REF at all times when the direct connection is used.)

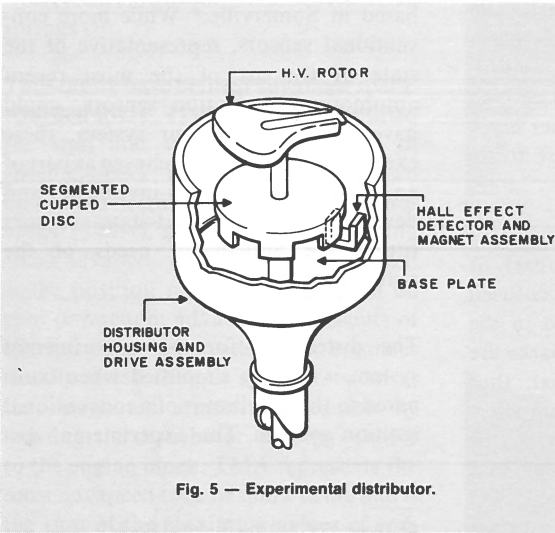


Fig. 5 — Experimental distributor.

The vacuum sensor connects to the normal "vacuum advance port" on the engine. Pressure (less than atmospheric—hence vacuum) variations cause a flexible diaphragm in the sensor to change position. A Hall transducer converts the diaphragm position to a voltage. This analog voltage controls the pulse width of a triggered monostable, providing two outputs that are a function of vacuum—the continuous analog voltage and the width of the monostable pulse.

Crankshaft rotation is measured by a variable-reluctance magnetic sensor in close proximity to the gear teeth on the outer circumference of the flywheel. The output from the sensor is approximately sinusoidal in shape, and as there are 164 teeth on the test vehicle's flywheel, the frequency is 164 cycles per crankshaft revolution. The sensor output is converted by the crankshaft electronics to a 328-cycle/crankshaft-revolution square wave of constant amplitude (5V). These electronics consist of a full-wave diode bridge, voltage clamping, amplification, and wave-shaping circuits. The frequency is doubled to enable its direct use in the spark calculation to a resolution of approximately 1 degree.

The ignition coil used is a standard high-energy coil, controlled by a high-energy electronic ignition module intended for operation with a breakerless distributor. Our only modification to the ignition module was the addition of an intermediate-stage dwell output. (Dwell is initiated by the ignition module electronics and this information is an input to the digital system.)

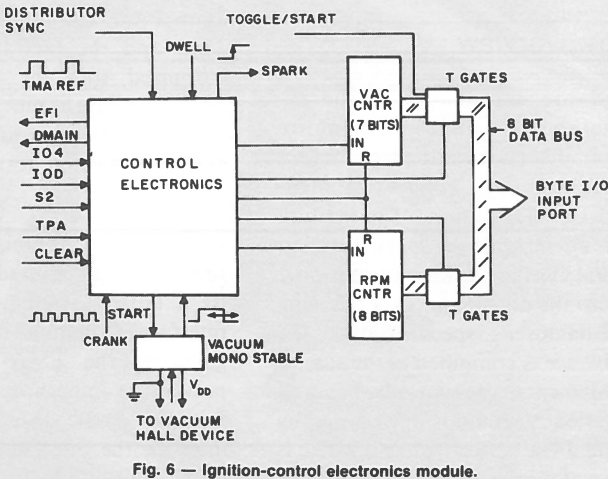


Fig. 6 — Ignition-control electronics module.

COSMAC interface

The experimental system was developed on the COSMAC Microkit, a hardware/software development system based on the RCA CDP1801 microprocessor.⁴ The interface circuitry unique to the ignition system resides on a 4- by 6-in breadboard that fits into one of the empty slots in the Microkit backplane. A block diagram of this module is shown in Fig. 6. The Microkit provides facilities not required for a final product; the next step in a product design would be to create a minimum-cost system that performs in a functionally identical manner. The interface described here uses the following Microkit facilities: microprocessor, RAM memory, PROM memory, address latch, I/O decode, and a byte I/O input port (interface to RAM memory).

The Microkit operates at the standard $V_{DD} = 5V$ and clock frequency of 1.95 MHz. This gives a machine cycle time, and TPA (CPU timing pulse) period, of a little over 4 μs , and an instruction fetch/execute time of a little over 8 μs . TPA, appropriately gated, drives both the VAC and RPM counters. The contents of the VAC counter (and the toggle/start switch position) are read into memory by issuing an I/O instruction (IOD) just after REF. The contents of the RPM counter are read into memory by DMA input cycles caused by crank square-wave transitions (82 per firing cycle). Since DMA cycles automatically increment COSMAC register R0, its position serves as a "tooth degree indicator." R0 is reset by program at each occurrence of TMA. The program in-

itiates spark issuing an I/O instruction (IO4) when the address in R0 matches a pre-calculated value, which is a function of the advance.

Program description

As can be seen in the functional flow chart of Fig. 4, the program operation is synchronized by the distributor waveform, which the program senses by testing an external flag (EF1). At REF, by which time spark has initiated, the VAC and RPM counts are read and the spark advance is computed, in tooth degrees. The program translates this to a count relative to TMA and waits for TMA. At TMA the two counters are reset and the vacuum monostable is started. When the address in R0 reaches the appropriate tooth degree count, spark is initiated and the program execution waits for REF to start the next cycle.

Both the vacuum and rpm advance computations use a table look-up and interpolation subroutine, as both advance components are nonlinear functions of VAC and RPM. The smooth advance curves shown in Fig. 7 are drawn through limit values specified by the manufacturer. Obviously, mechanically-caused advance is not expected to be very precise. The discrete points within the curves correspond to look-up table values stored in the program. Each of the tables actually has 16 entries, but since the abscissas are counts that are not scaled to fit the bounds exactly, some points spill beyond the limits shown (and are never actually used). The table look-up routine linearly interpolates values between the points shown to an accuracy of one part in eight. On average, about 60 instruction executions are required to return an interpolated advance value, given a vacuum or rpm value. Thus the computational portion of program, from TDC to REF, requires roughly 1 ms of processing time. Even at 4000 rpm, roughly 2.5 ms are available in this interval, so the computing load on the microprocessor is light.

The program itself occupies 192 bytes of memory storage, broken down as follows:

	Bytes	%
Main loop	71	37
Look-up subroutine	64	33
Initialization and start-up	25	13
Look-up table data	32	17
	192	100

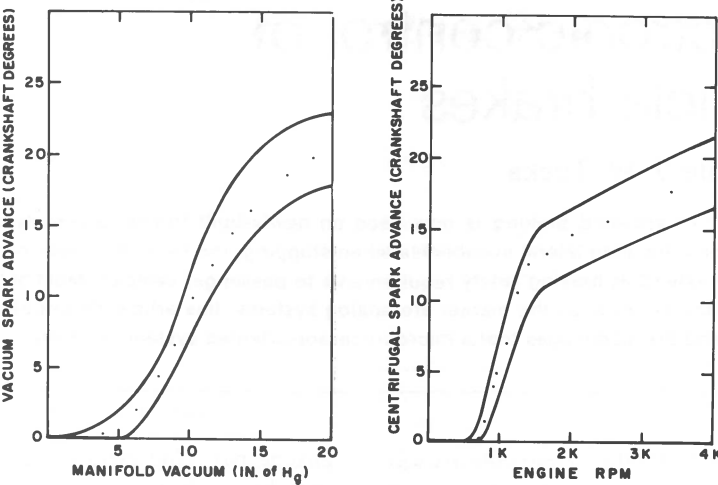


Fig. 7 — Manufacturer's specifications for mechanical control of ignition. Dots indicate stored-program data.

It uses more RAM than it needs to, because it is provided in the Microkit. Since only 7 of the 16 available COSMAC scratch-pad registers (2 bytes each) are used, only a single byte of RAM for arithmetic and input/output would be required.

System performance

When this microcomputer-controlled spark advance system was first installed on our test vehicle we adjusted the distributor position such that default firing, or spark firing without vacuum or RPM advance, would occur at the desired time. We used a high-voltage stroboscopic timing light that we had used on this and other engines many times. This time the timing marks on the rotating damper, illuminated by the timing light, were much sharper and more clearly defined than we had seen before on this or other vehicles. We had expected increased accuracy and improved transient response from our system, and this visible demonstration of more accurate repeatability was the first of many on-the-car confirmations of our expectations. The vacuum and rpm components of spark advance have been measured individually, and in combination, on the operating engine. (The timing light was clamped in position for accuracy, and the standard timing scale (2° graduations) replaced with a temporary scale with accurately marked 1° graduations. The measured advance components were within $\pm 1^\circ$ of the data points in Fig. 7, with the exception that approximately 2° of spark advance results from manifold vacuums between 3 and 4 inches of Hg.

Our station wagon, with its microcomputer-controlled spark advance system, has been driven in dense traffic and on open interstate roads, and in temperatures varying from summer to winter in New Jersey. The system has been subjected to a variety of intra- and extra-vehicular electrical noise, including the repeated cycling of the vehicle's air-conditioner clutch, normal vehicle controls (e.g., blower, turn signals, etc.), a 550-W inverter, and the car's CB transceiver. The vehicle has been driven in the vicinity of rural electric fences and rf transmission sites, and has been subjected to the radiated noise from SCR-controlled power tools. During all of this testing we have not experienced a failure or detected any irregularities.

The "driveability" of a vehicle is a difficult term to define and impossible to measure in an engineering sense. However, the change in driveability of the test car with the experimental spark advance system, as compared to the same car with a well-maintained, freshly-timed conventional system, is noticeable, with an apparent improvement in smoothness and response. The vehicle's responsiveness, when the 302-cubic-inch engine is operating under microcomputer-controlled spark, is more readily felt than described.

References

- Norris, J.C.; "Delcotron breakerless transistor ignition system," SAE paper 617B (Jan 1963).
- Carlson, D. W.; and Doelp, W. L.; "A successful electronic ignition system through fundamental problem analysis," SAE paper 740154 (Feb 1974).
- Ford Shop Manual, 1969, Vol. 2.
- MPM-103, "COSMAC Microkit Operator's Manual," RCA, SSD, Somerville, N.J.

Electronic control of vehicle brakes

W.R. Lile|J. W. Tuska

Anti-lock or anti-skid braking is now used on newly-built trucks to comply with the regulations that limit lateral instabilities when stopping; the Federal Government could possibly extend its braking safety requirements to passenger vehicles. Most of the anti-skid controllers now on the market are analog systems; this article describes a digital system and the advantages that a microprocessor-oriented system can have.

IN THE 1950's the aircraft industry was the first to use electronics to prevent wheel lock-up during hard braking. The need to develop a better brake system became important as pilots had to exercise judicious restraint in braking to minimize tire blow-out, maintain lateral stability, and still stop on the runway.

Several years later, in the 1960's, automobile manufacturers began studies to improve braking systems for road vehicles. They realized—as had the aircraft industry—that a locked wheel causes a loss of lateral stability. Chrysler began to convert their research on anti-skid braking into a production system in 1966;¹ the final product was marketed in 1971. Other manufacturers soon followed suit.

Safety has always been difficult to sell to a "style conscious" public. Why should

anyone pay \$200 extra for a device to keep his car from skidding? How can it protect the average driver? Is it fail-safe and always advantageous? These questions are still being debated as the public slowly becomes safety conscious.

Federal regulation

We are familiar with the federal government's involvement with automotive design when it imposed pollution standards on the internal combustion engine. There was some give and take, and the standards were postponed and even modified when lawmakers were confronted with realistic engineering problems. The net thrust of the standards remained intact, however; the designers worked and met the challenge, and the price of cars went up.

A similar situation now exists with FMVSS 121 (Federal Motor Vehicle Safety Standard No. 121), which became effective September 1, 1974. In essence, this standard requires all new trucks to remain within a certain width lane while undergoing panic or violent braking. FMVSS 121 does not require anti-lock devices, but its lateral and directional stability requirements imply their use. The object of the government edict was safety, but, again, postponements and modifications have changed FMVSS 121 from its original form and created new deadlines. As yet, no standard exists that requires similar braking restrictions on passenger vehicles. If one is issued, there will be a large market for anti-skid controllers, but cost considerations will otherwise keep the passenger-car market at a minimum.

Skid and slip

All anti-skid controllers anticipate wheel lock-up and momentarily release the brakes, then reapply them when the wheel speed recovers sufficiently. One manufacturer of anti-skid controllers makes a digital system,² while all the others use an analog computer with operational amplifiers and discrete components. The use of microprocessors may yield a significant advantage over these existing systems. Let us next examine what is meant by "skid" and exactly what an "anti-skid" system does.

A freely-rolling wheel permits maximum directional control of a vehicle. If the wheel becomes locked and skids, the vehicle loses ability to remain on course.³ This occurs if the driver vigorously, or, in panic, applies the brakes. When a vehicle is making the transition to a stop, its kinetic energy must be expended somewhere. In a controlled stop, most of the kinetic energy is dissipated in the brake linings and drums as heat. Some is lost in the tire-road interface. However, if the brakes are locked, no energy is lost to the brakes and all the kinetic energy of the vehicle is lost in the tire-road system. Six such stops from 60 mi/h can wear out a new set of tires.¹

Barely touching the brakes will produce a very slow stop and dissipate almost all the energy in the brake system. Between these extremes lies an optimum combination of short stopping distance and good lateral directional stability.

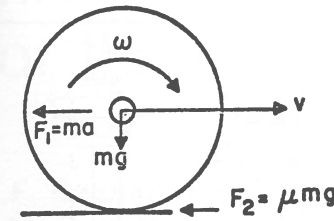


Fig. 1 — Simplified tire-road interface forces.

Fig. 1 illustrates the forces at the tire-road surface interface. The retarding force is $F_1 = ma$ where m is the vehicle mass and a is the deceleration. This is equal to the frictional force $F_2 = \mu mg$, where μ is the dimensionless coefficient of friction and g is the gravitational acceleration constant. The maximum deceleration of the vehicle is 32 ft/s^2 , obtained when $\mu=1$.

A free-rolling wheel in the vehicle system indicates the true vehicle speed and deceleration. Best stopping occurs, however, when the braked wheels actually slip. Slip is expressed as $S = (\omega_0 - \omega) / \omega_0$, where ω is the angular velocity of the braked wheel and ω_0 is the angular velocity of its free-rolling counterpart.⁴ For example, if a vehicle were going 50 mi/h according to the free-rolling wheel and the braked wheel indicated only 40 mi/h, then 20% slip would exist.

Fig. 2 shows the relationship between the retarding (μ_1) and lateral (μ_2) characteristics as a function of wheel slip.⁵ The lateral stability coefficient μ_2 is at a maximum at zero slip and zero at 100% slip. The retarding force characteristic μ_1 is at a maximum at about 20% slip on a normal road surface. Brake controllers then should strive to maintain this optimum slip. However, the problem is not simple, as the coefficient curves change with varying road surfaces.

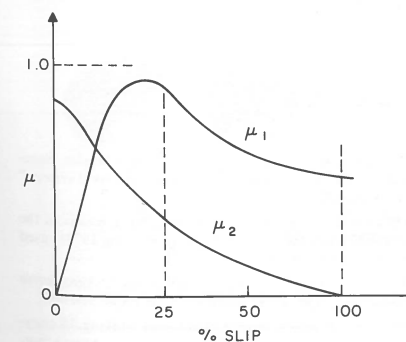


Fig. 2 — Retarding (μ_1) and lateral (μ_2) characteristics vs slip.

For example, an ice-covered road has its peak retarding force at about 10% slip.

A general indication of wheel lockup occurs at about 1.2 to 1.8 g's wheel deceleration, indicated by an angular velocity sensor on the wheel. This threshold is empirical, but since the vehicle cannot attain this deceleration rate, an indication greater than 1g implies incipient lockup.

Wheel angular velocity sensors

The devices that sense wheel rotation must be rugged, reliable and inexpensive; variable-reluctance sensors are the most common in use today. The pick-up coil assembly is mounted on the axle and a slotted disc or rotor is mounted on the wheel. The output is Hz/mi/h; a change in frequency indicates that an acceleration is taking place. These sensors have a lower speed limit of about 5 mi/h, due to the dependence of voltage output on speed, but are considered to be the most reliable component of the system.⁴

Computer

Wheel angular velocity information is supplied to a computer, which then decides if a lockup is imminent and issues a brake release signal if needed. Once the brake is released, the computer monitors the wheel sensor to determine when to reapply the brakes.

Brake pressure modulator

If the computer determines that the wheel is going to lock up, it issues a signal to a solenoid-actuated modulator, which closes the brake fluid line to that wheel. The modulator also provides an expansion chamber for the fluid on the wheel

side of the valve, thereby reducing the pressure and diminishing the braking force.

Stopping the vehicle

Fig. 3 illustrates both the angular velocity of the brake-modulated wheel and the true vehicle speed as a function of time.

The slopes are indicative of both deceleration and acceleration. Fig. 4 depicts an ideal smooth stop, where there is precise control of the braked wheel's speed. Slip is monitored in this case, rather than acceleration or wheel speed. One commercial manufacturer² of truck anti-lock systems incorporates a longitudinally-oriented accelerometer as an additional sensor. The accelerometer output signal is integrated from the onset of braking to determine true vehicle speed. Knowing true vehicle speed and the braked wheel's angular velocity, slip can be controlled. In general, though, the cost of independent vehicle-true-speed measuring devices outweigh their advantages, so manufacturers do not incorporate them. Rather, the speed at the onset of lock-up is stored and further wheel angular velocities are compared with it until the next lock-up is recognized.

Looking again at Fig. 3, the slope of the braked wheel's angular velocity is related to the coefficient of friction μ of the tire-road interface. The small the μ , the faster lock-up can occur; the greater the μ , the faster recovery can occur. Also, the greater the μ , the faster the vehicle decelerates. Since μ varies with road conditions, using its instantaneous value rather than an assumed one would help determine brake release and reapplication times more accurately. Existing brake control systems, to the authors' knowledge, do not attempt to measure μ .

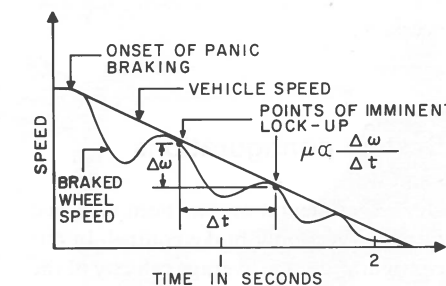


Fig. 3 — Vehicle's braked-wheel speed in a controlled system.

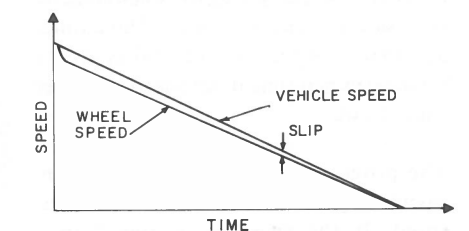


Fig. 4 — Nearly ideal stopping at optimum slip.

J. Tuska (left) and W. Lile examining electronic braking data in the RCA Laboratories test vehicle.



William R. Lile, LSI Systems Design, RCA Laboratories, Princeton, N.J., received the BS in Electrical Engineering in 1959 from Mississippi State University. He attended the University of Pennsylvania on the RCA Graduate Study Program and received the MSEE there in 1962. Mr. Lile has also completed additional graduate courses beyond the MSEE requirement and is a licensed Professional Engineer. After joining RCA's Electronic Data Processing Division in Camden, N.J., in 1959, Mr. Lile worked on logic design and worst-case analysis of advanced digital equipment. He was responsible for the electronics and test instrumentation in the superconducting memory effort. He has been a Project Engineer for LSI memory arrays and has published papers in this field. At present he is in the LSI Systems group, engaged in the application of integrated-circuit technology, including microprocessors, to automotive and other vehicle systems. Mr. Lile is a member of Tau Beta Pi and Eta Kappa Nu.

James Tuska's biography appears with his other paper in this issue.

Final manuscript received April 25, 1976.

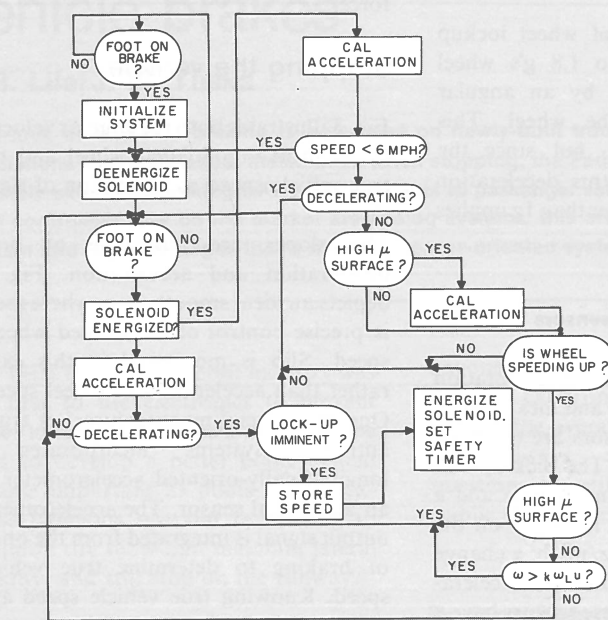


Fig. 5 — Electronic brake control algorithm

Foot-on-the-brake flowchart

The flow diagram of Fig. 5 illustrates a basic algorithm that a digital microcomputer might use for controlling vehicle brakes. This algorithm assumes one sensor and one-wheel control. In practice this sensor could be a pinion (propeller) shaft sensor, and the brake line pressure modulator could be in common with both rear wheels.

An explanation of the flowchart can begin at the "foot on brake?" box. Once the system recognizes that the brakes are being applied, all registers are initialized, and the solenoid that releases brake pressure when activated is de-energized. The acceleration (deceleration) is calculated next ("cal acceleration" box). If it is great enough that there is a likelihood of the wheel's locking up, then the speed is stored as ω_L , the brake modulator relieves the pressure in the brake line by energizing the solenoid, and the safety timer is set. This timer automatically releases the brakes after a predetermined time if the microcomputer should fail.

The program then continues in a loop, calculating acceleration and vehicle speed. If the speed is greater than 6 mi/h, the program determines the point of positive acceleration (the wheel is now speeding up, as the brakes are off). If the

road surface has a high μ , the wheel speeds up rapidly. To assure this is actually the case and not just a bounce, a second acceleration calculation is made. If the wheel is still speeding up, and the interface is a high- μ one and the solenoid is de-energized, the brakes are now reapplied. When the speed falls below 6 mi/h, the solenoid is de-energized and there are no further calculations until the next time the brakes are applied.

If the road-tire interface is slippery, the wheel will recover more slowly. If this is the case, the wheel speed must be monitored until it exceeds the release value ω_L . At this time the brakes are reapplied. If this low- μ determination were not made, the positive acceleration criteria (high- μ) might not allow the reaplication of the brakes, since recovery is slow on slippery surfaces.

The cycle is repeated as many times as necessary.

System configurations

Our experimentation has been confined to rear-wheel-only brake control. In our configuration, the average velocity of the wheels is sensed by a pinion shaft sensor, and a single modulator controls the brake pressure to the rear wheels. This is shown

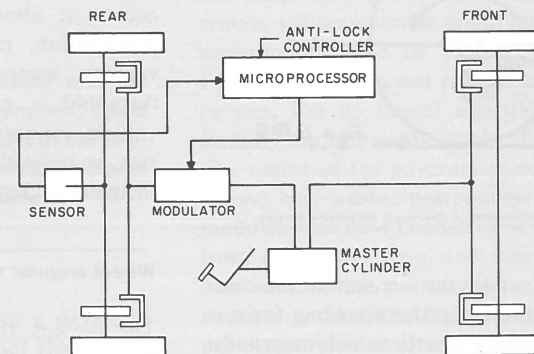


Fig. 6 — Two-wheel, rear-axle control, propellor-shaft sensor system.

schematically in Fig. 6. Other system configurations have been proposed,³ each with its own advantages, disadvantages and cost/benefit ratio.

Conclusions

Discrete-component analog controllers dominate the anti-skid brake market now, but the advantages of stored program controllers are becoming evident. Their processing capabilities can handle multiple axles and sensors simultaneously, as well as take instantaneous variations in tire-road conditions into account. As tests proceed, we expect to further modify and enhance our basic algorithm, investigate effects of multi-wheel, multi- μ surfaces and demonstrate how a microprocessor can be used to better advantage in this application.

References

1. Douglas, J.W. and Schafer, T.C.; "The Chrysler 'Sure-Brake' — the first production four-wheel anti-skid system," SAE Paper 710248.
2. "FMVSS 121 — air brake systems: what it means to the truck builder," *Automotive Engineering* (Aug 1973) based on SAE Paper 730698.
3. "Anti-lock brake system configurations," *Automotive Engineering*, (Feb 1976) based on SAE Paper 760348.
4. Frait, J.S.; "Practical aspects of anti-lock braking," Kelsey-Hayes Research and Development Center, Ann Arbor, Mich.
5. "A review of anti-skid braking," *Automotive Engineering* (July 1975) based on SAE Papers 690213, 710248, 741083.

Microcomputer control for the car of the future

E.F. Belohoubek
J.M. Cusack
J.J. Risko
J. Rosen

Several years from now, \$200 may buy a radar/display system that will make your driving safer.

Erwin Belohoubek is Head of the Microwave Circuits Technology Group at RCA Laboratories, a group working on microwave hybrid integrated circuits. In this capacity, he is responsible for the development of passive and active MIC circuits, including high-power transistor amplifiers, multipliers, linear bipolar and FET amplifiers, electron-beam semiconductor amplifiers, circulators and limiters, and small microwave subsystems.

Jerome Rosen has been engaged in the research and development of microwave devices and subsystems in the frequency range of 30 MHz to 40 GHz since 1957. His current automotive-related assignments include a self-mixing X-band Doppler radar, the electronic-signpost Automatic Vehicle Monitoring system, and the electronic tag of the microwave Automatic Vehicle Identification system.

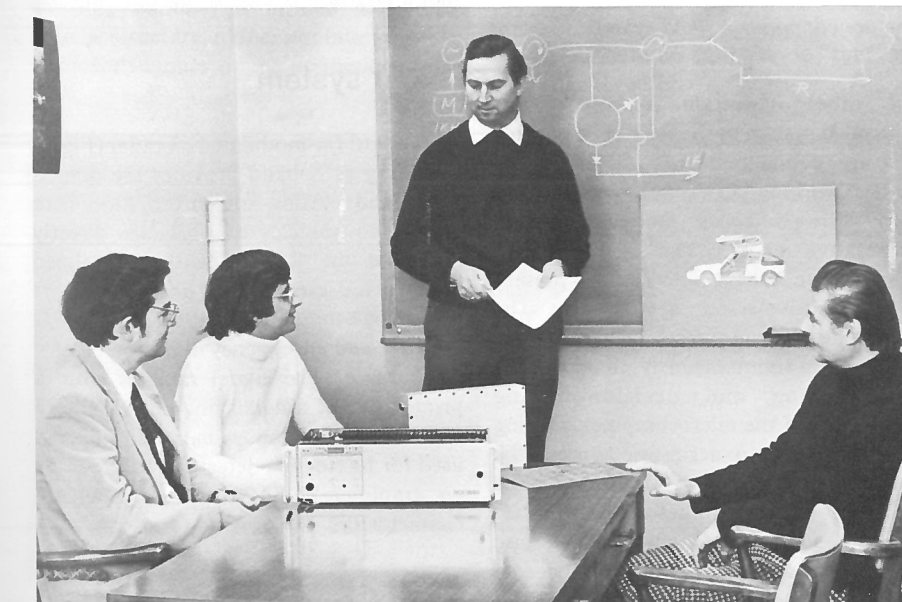
Joseph Cusack has principally worked in the area of minicomputer automation of microwave and thermal measurements since he joined RCA in 1972. In 1974-1975, he participated in the development of a computerized satellite-checkout station for the RCA Satcom satellites. Recently, he has been investigating the potential of microprocessors in microwave applications.

John Risko joined the Microwave Research Laboratory in 1962, working on GaAs and GaSb tunnel diodes and varactor diodes. He has since worked on silicon devices, including IMPATTs, TRAPATTs, and more recently BARITT structures. Mr. Risko received a 1972 David Sarnoff Research Laboratories Achievement Award for "team effort leading to the development of S-band Trapatt Amplifiers."

Authors (left to right) **Risko**, **Cusack**, **Belohoubek**, and **Rosen**

Contact them at:

Microwave Circuits Technology, RCA Laboratories, Princeton, N.J., Ext. 2629



Until recently, car manufacturers have generally been slow to introduce electronic advances to the automobile. However, this situation is changing, and a definite trend toward using automotive electronic systems is becoming evident. This is especially true with the advent of the microprocessor, which has opened up a wide range of possibilities for controlling relatively complex electronic systems in a cost-effective manner. Two examples of automotive microprocessors already in use, or shortly to reach the market, are anti-skid systems¹ and the various forms of electronic fuel control² used for improved economy and low exhaust emission. Looking toward additional applications in the future, the Microwave Technology Center at RCA Laboratories is presently developing a microprocessor-controlled information and safety system for a Research Safety Vehicle (RSV), sponsored by the U.S. Department of Transportation.*

The major objective of this program is to develop a safety car in the 2000-lb class for the mid-1980s. Fig. 1 shows a preliminary mechanical mock-up of the car, which will include many innovations such as a lightweight, foam-filled frame with plastic body-glove covering, an air-bag restraint system designed to provide protection to passengers in frontal crashes up to 80 km/h, an anti-skid braking system, pedestrian impact protection, a special front-end and side-frame structure designed to absorb energy in high-speed crashes, etc. RCA's contributions to the program are a microprocessor-controlled noncooperative radar and an electronic dashboard display. The radar and the display are interfaced with a number of safety-related sensors throughout the car

*The RSV is being developed by Minicars, Inc. of Goleta, Cal., with RCA as the electronics subcontractor.

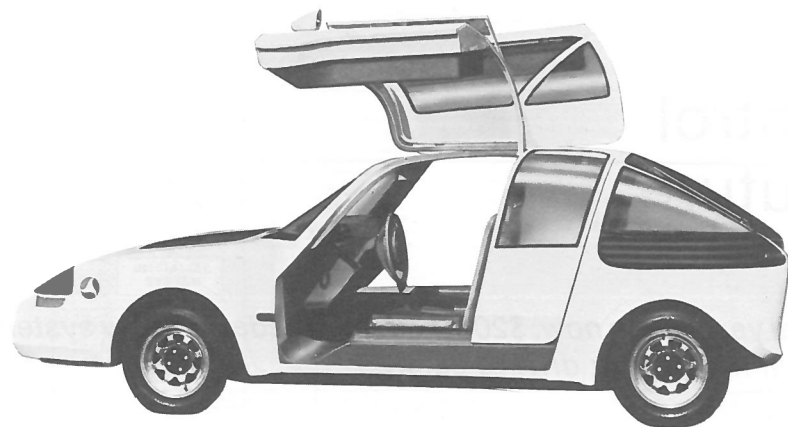


Fig. 1
Research Safety Vehicle, shown in mock-up form here, will have a number of safety features besides the electronics system that RCA is providing.

that determine unsafe operating conditions, provide warnings, and can control the RSV's braking and acceleration via the microprocessor.

Work on the RSV began in the middle of 1975; the present development period is aimed at demonstrating the feasibility of the various concepts in an experimental vehicle. The Dept. of Transportation expects prototype cars to be fabricated in 1977-78, with extensive road-testing to follow. Since the present electronic subsystem is designed mainly to show the feasibility of its various functions, a substantial effort will still be required, especially in the radar area, to refine the individual components for actual use in cars.

Outline of the two systems

Fig. 2 is a block diagram of the electronic subsystem for the RSV. There are two separate microcomputer systems in the developmental car—one for the radar, and one for the central display and its associated sensors. This arrangement provides maximum flexibility in testing and changing algorithms. However, in the final version, most of the computer functions will be designed into a few LSI chips, leading to greatly reduced size and cost. Such a central microprocessor interfacing with a number of sensors and an electronic display opens a nearly unlimited number of special convenience- and safety-related functions; we explore only a moderate number in the present RSV.

The dashboard display is clutter-free and easy to read.

The electronic dashboard display is the information center of the car, displaying speed, fuel level, trip mileage, rpm, and

other relevant information in easy-to-read, luminescent-orange alphanumerics. Clutter is greatly reduced, since the display provides only the most important driving-related information. All warnings about car-component malfunctions—low oil pressure, high water temperature, or anti-skid system inoperative, for example—appear as three-second override messages on the display and are repeated in thirty-second intervals until the malfunctions are corrected. The regular driving information remains on display between the warning messages. Since the single-line display is located right in front of the driver, directly below the normal line of sight onto the road, it is nearly impossible to overlook the flashing of an emergency or warning message. At the same time, the display greatly simplifies the dashboard, so the available space can be used for other convenience features, such as CB radio, tape cartridge player, etc.

The noncooperative radar system does not use "tags" or reflectors on other cars.

The other important microprocessor-controlled system in the RSV is a forward-looking noncooperative fm/cw radar. Earlier work at RCA Laboratories had been devoted to a second-harmonic, cooperative radar system³ that provided much of the background for the present noncooperative system development. The new radar unit has several functions that it can perform simultaneously, thanks to the programming and decision-making capabilities of the microcomputer. During highway driving under cruise control, the radar monitors the space ahead, and if the RSV approaches another car too closely the microcomputer deactivates the cruise control and automatically maintains safe headway. This is done by monitoring the

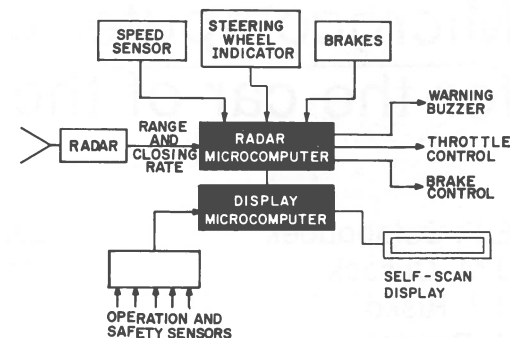


Fig. 2
Electronics system is divided into two separate microcomputer systems for ease in testing and changing algorithms.

range and closing rate with respect to the vehicle in front. As soon as the path ahead is clear, the cruise control goes back into operation. The radar also provides audible warnings for obstacles and other cars up to 30 meters ahead, which is especially important when driving in fog or with impaired visibility.

The last and possibly most important function of the radar system is to apply the brakes automatically in situations where a serious collision appears to be unavoidable. This braking function is automatically disabled if the driver takes evasive action or applies the brakes himself. The system design gives high priority to avoiding false alarms that could trigger accidents or cause the driver to lose faith in the system. Therefore, the automatic braking will be activated rarely—only after a series of measurements and sensor indications clearly determine that a collision is indeed unavoidable.

Radar system

A standard fm-modulated cw radar (Fig. 3) operating at X-band provides the desired range and closing-rate information with respect to vehicles or obstacles directly ahead of the RSV. The varactor-tuned transferred-electron oscillator is frequency-modulated at a sweep rate of $f_m = 1$ kHz and a frequency deviation of $\Delta f = \pm 25$ MHz. The signal radiates from a printed-circuit antenna mounted under a radome in the hood. A balanced mixer is used for homodyne detection, followed by an amplifier chain with a shaped-gain characteristic.

Final manuscript received November 12, 1976.

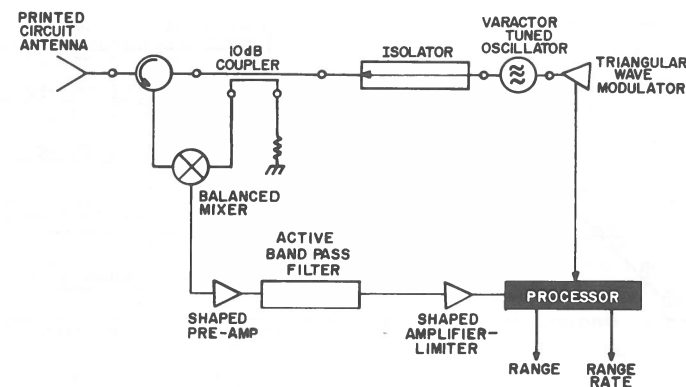


Fig. 3
Fundamental-frequency radar produces range and range-rate outputs that are used for making warning/braking and headway decisions.

The antenna consists of 128 fan-shaped dipoles and a feed structure printed on both sides of a Duroid circuit board. To reduce the number of possible false alarms that otherwise would arise from parked cars, overpasses, and road signs, the antenna beamwidth and both the vertical and horizontal side-lobes must be very small. Consequently, the beamwidth of the printed antenna is approximately 5° in azimuth and 10° in elevation, and a Chebyshev current-taper⁴ is used to excite the individual antenna elements to reduce the side-lobes. The side-lobe levels listed in Table I represent the performance of the antenna with such a taper. Adding shielding and damping material a few inches ahead of the antenna reduced both horizontal and vertical side-lobes to below -20 dB, which proved to be sufficient for eliminating false alarms caused by the antenna side-lobe responses. Since the damping material is located under the radome structure, it does not interfere with the aerodynamics or the aesthetics of the RSV.

Range and range-rate information is derived by processing the i.f. signal that appears at the mixer output. For an fm/cw radar with a triangular fm modulation the beat frequency is of the form:⁵

$$f_B = |8(R/c)\Delta f f_m \pm (2\dot{R}/c)f_o| = |f_R \pm f_D| \quad (1)$$

where R is the range, $\dot{R}$ the range rate, f_o the carrier frequency, and c the velocity of light. For $\dot{R} > 0$, the plus sign in Eq. 1 gives the beat frequency during the up-swing of the modulation, while the minus sign corresponds to the down-swing. In Eq. 1, f_R is often called the range frequency, and f_D the doppler frequency. For the parameters chosen in our system a range of 1 m corresponds to $f_R = 667$ Hz and a range rate of 1 m/s to $f_D = 70$ Hz. By measuring the beat frequency during both the up- and down-swings of the modulation, range and closing rate can be determined separately:

$$R = (c/16 \Delta f f_m)(f_{Bup} + f_{Bdown}) \quad (2)$$

$$\dot{R} = (c/4f_o)(f_{Bup} - f_{Bdown}) \quad (3)$$

Table I
Performance specifications for radar system show small size, accurate range and range rate, and low gain at the side-lobes to reduce potential false alarms.

Frequency:	10.575 GHz
Stability:	within 1% from -50° to $+60^\circ$ C
Power output:	27 mW
Frequency deviation:	± 25 MHz
Modulation rate:	1 kHz
Size:	$32 \times 19 \times 5$ cm
Power consumption:	6 W
Range:	6 to $30 \text{ m} \pm 0.2 \text{ m}$
Range rate:	0 to $160 \text{ km/h} \pm 10 \text{ km/h}$
Antenna gain:	25 dB
Side-lobes horizontal:	-18 dB
Side-lobes vertical:	-15 dB

The microprocessor calculates the beat frequency, f_B , by using a 1.95-MHz clock to count the time between successive zero-crossings. The calculation is performed over several modulation periods and averaged; this procedure guarantees a much higher accuracy and lower granularity than the conventional method of counting zero-crossings.

During our early experiments, we used the range information obtained in this form and differentiated it with respect to time to determine the closing rate. Provided sufficient time, on the order of a few tenths of a second, is available, this computation is fairly accurate and can be used for the headway-control algorithm. To initiate emergency braking, however, the response-time must be much shorter, so the doppler frequency must be extracted directly from the beat frequency. Since the doppler frequency is the result of subtracting two relatively large numbers from each other, the accuracy requirements for the beat-frequency measurements are more severe than for range-measurement alone. The present system therefore measures f_D to an accuracy of 12 bits, which requires approximately 80 ms for one averaged range and range-rate reading with the 8-bit microprocessor. If a special-hardware multiplier were used, this time could be reduced to about 30 ms.

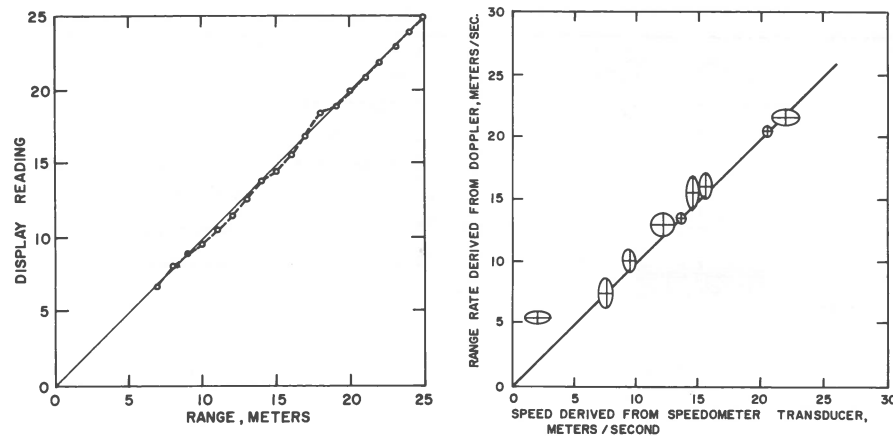
Microprocessor radar algorithm

An RCA COSMAC development system (Microkit) using the CDP1801 microprocessor performs the calculations and decision-making for the radar in two basic areas—warning/braking and maintaining a safe headway distance. The system converts radar data from a digital interface into range and range rate (closing velocity). It then takes this information and combines it with other sensor inputs and decision criteria to generate an audible warning or an output that will operate the car's brakes when necessary. The system determines a safe headway distance with respect to vehicles ahead by using the range and range-rate information together with car speed.

Range is derived from the radar interface input according to the equation

$$R = K_R N (1/COUNT_{up} + 1/COUNT_{down}) \quad (4)$$

where K_R is a lumped constant depending on the radar parameters f_m , Δf , and samp-



Figs. 4a (left) and 4b

System is accurate to ± 0.2 m in range (left) and ± 10 km/h in range rate. Transducer inaccuracy at very low speed produced the off-line data point in the range-rate graph.

ling frequency f_s . N is the (programmable) number of beat-frequency periods over which the measurement is taken. $COUNT_{up}$ is the number of periods of the sample frequency f_s that occur over N periods of the beat frequency during the first half of the modulation cycle, and $COUNT_{down}$ is similarly defined for the second half of the modulation cycle. Range rate is determined by measuring the difference between $1/COUNT_{up}$ and $1/COUNT_{down}$ and using another calibration constant, K_D , in place of K_R . Special threshold circuits provided in the hardware interface between the radar and the Microkit inhibit the range measurement when the signal drops below a given level any time during the measurement cycle.

Checking the radar's accuracy

The accuracy of the measurement system was checked by measuring the range to a stationary corner reflector having dimensions equivalent to a 10-m^2 radar cross-section, corresponding to that of a medium-sized car. Fig. 4a shows stationary range measurement data obtained from such a well-defined target; the range accuracy is typically within ± 0.2 m. Range-rate measurements were performed by driving the car at constant speed towards a tape-supported corner reflector that was pushed out of the way at impact. This relatively simple test permitted taking data up to speeds of 80 km/h, as shown in Fig. 4b. Under actual driving conditions, the accuracy of range and range-rate informa-

tion could be substantially lower because of changes in the effective reflection point on the target vehicle that are caused by slight steering-angle variations, up-and-down movements of the car, etc.

As the range and range rate are continually updated, they are used together with other sensor inputs to perform a number of control functions. The present system provides the necessary signal outputs for warning the driver about obstacles ahead and for automatic braking in case of an unavoidable collision.

Avoiding false alarms

In most cases, an alert driver will be able to avoid a collision by evasive action or braking. For this reason, the system will not generate a braking output if the driver is already braking or if he is turning the steering wheel hard enough to avoid an accident. This is done by having the microprocessor act as a filter that sharply limits the conditions under which the braking output will be generated. Although this reduces the benefits of the system somewhat, it does prevent unnecessary braking.

Also, in a normal driving situation, a great number of nondangerous targets may appear in the line of sight of the radar. The system can reject many of these targets by ignoring all the ones farther away than some maximum range, nominally 25 m. In addition, a minimum speed threshold inhibits the processor from initiating braking if the vehicle speed drops below some lower

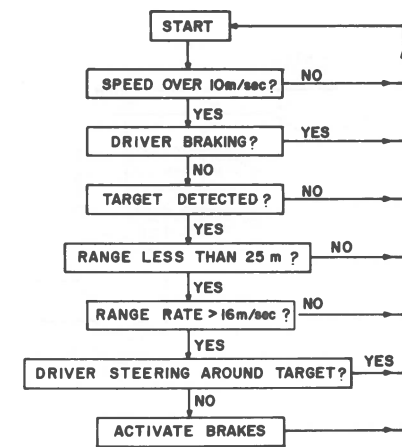


Fig. 5
Braking system avoids false alarms by checking six conditions before activating brakes. This avoids false targets that take place during slow-speed maneuvering and also gives driver control of the car if he is trying to avoid an accident.

limit, nominally 36 km/h. This restriction again reduces the system benefits somewhat by disallowing operation for very-low-speed collisions, but it also rejects a number of false targets that could occur during slow-speed maneuvering, as in a parking lot.

Fig. 5 shows a sample flowchart for the braking-algorithm software. Since using a microprocessor permits radical changes in system performance characteristics by changing software, the microprocessor acts as a useful development tool as well as a sound way to implement the final logic design. To date we have experimented with linear expressions of the form

$$K_1V + K_2\dot{R} + K_3R > 0 \quad (5)$$

in order to generate outputs for an audible warning and collision-mitigation braking. Such linear equations have traditionally been used in this field because they are easily implemented with analog hardware. The microprocessor, however, can also handle nonlinear control equations where it becomes necessary.

The radar system also maintains a safe headway with respect to vehicles ahead of the RSV. For this purpose, range rate is derived by differentiating range with respect to time, because adjustments in headway have a relatively long time constant (roughly a second) and require a higher accuracy in range rate than the emergency braking algorithm does. This more accurate range is important to keep

the feedback control loop for the car-following algorithm⁷ stable and free from short-term braking or acceleration impulses. In the present system, the outputs from the microprocessor to the car's throttle and brakes are not yet interfaced, but are shown as display messages—"speed up" or "slow down." The actual interface with the car is part of the next phase of the RSV program.

Testing performance on the highway

A series of road tests on both regular highways and winding country roads was performed to check the operational capabilities of the radar system. In the cruise-control mode, modified for safe headway keeping, the radar demonstrated excellent tracking capabilities with respect to cars ahead. The system normally tries to keep a predetermined cruising speed (presently, by showing the proper "speed up" or "slow down" commands), and if the car approaches another vehicle within 25 m it keeps a speed-dependent safe headway with respect to this vehicle. The radar in these tests was not disturbed by any false readings from oncoming traffic, passing bridges, road signs, etc. One of the major difficulties encountered was that although most vehicles have fairly large radar cross-sections, some cars, especially those with strongly slanted backs, provide only little reflected power, which leads to temporary signal dropouts.

The algorithm for automatic brake application was tested by driving the car at high speed through a stationary expendable corner reflector and monitoring distance from the target at which the brake signal was initiated. For speeds up to 100 km/h, the maximum that could be achieved on our test range, the brake signal occurred correctly within the expected range—10 to 25 m before the target. Using the same algorithm, threshold settings, and sensitivity, the radar did not produce any brake signal outputs during regular highway driving, indicating essential freedom from "false alarms." Such tests will have to be performed, however, on a much better controlled basis and for prolonged periods of time to truly optimize the braking algorithm for the wide variety of possible road and accident situations it may encounter.

Performance and safety sensors

A second COSMAC development system monitors vehicle performance and various safety-related sensors. The sensor status is shown on a Burroughs 32-character, single-line, self-scan display. Fig. 6 is a block diagram of the display-microprocessor system with its associated ROM and RAM memories.

The binary data interface has 24 switch closure inputs to buffers or inverters that are either "on" or "off" (12V or 0V). The status of the group-selected buffers or

inverters is read by turning on a transmission gate. The switch-closure sensors monitor hand-brake status, brake-fluid level, whether the doors are closed, and other similar, simple functions. The steering-wheel interface is a one-shot oscillator whose period of oscillation is controlled by the resistance of a potentiometer that senses the movement of the steering-wheel shaft. An analog/digital converter with a multiplexer digitizes analog inputs such as water temperature, fuel level, and oil pressure.

Two other sensors provide pulse trains whose frequency is proportional to fuel flow and engine rpm, respectively. The pulse trains clock counters, which are read at intervals determined by the software; the data is then converted into a form suitable for display.

The velocity-monitoring circuitry is an example of typical sensor interface card. It determines the car velocity by means of a generator attached to the RSV's speedometer cable. The generator's output frequency is proportional to the angular velocity of the speedometer cable, and when it is applied to an op-amp (CA3401) that is driven into saturation, it provides a squarewave input to an inverter. The resulting signal clocks a counter that in turn is read by activating a bilateral switch. The system software interprets the counter reading and displays the speed in km/h.

Central information display

An electronic display system presents the information pertaining to vehicle performance and safety devices to the driver. A Burroughs gas-plasma, single-line, 32-character "self-scan" display was selected as the best commercially available system for automotive applications at this time; Fig. 7 shows how the display appears to the driver. The "self-scan" unit uses x-y addressing and a 3-phase driver system to propagate a gas plasma the length of the display. The unit comes complete with memory for refresh and a character generator with a repertoire of 128 different characters. A dc-dc converter boosts the car's 12 V to 250 V to provide the operating voltage for the display.

The driver of the RSV can choose between two normal modes of information display, as Fig. 7 illustrates. One mode shows fuel level, rpm, and an analog representation of speed in the form of a horizontal bar graph. The other display mode indicates trip mileage, time, fuel economy, water

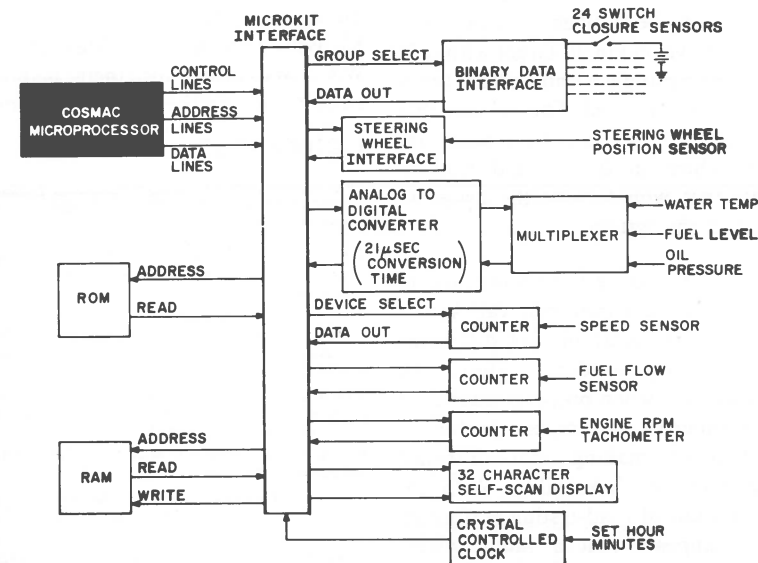


Fig. 6
Improved display system unifies the numerous bits of safety and performance information that are spread across today's dashboard.

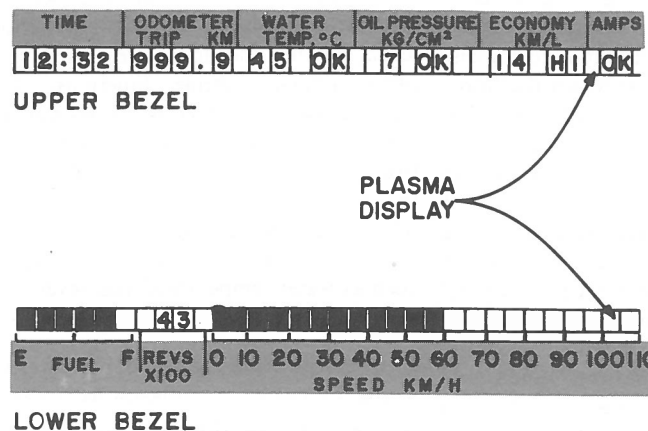


Fig. 7
Driver's view of the self-scanning display shows how above-the-dash location keeps driver's eyes close to the road while checking display information. Driver has choice of two display modes, with emergency messages overriding in case of a malfunction. Display is both digital and bar-graph in form.

temperature, oil pressure, and battery status.

In case one of the sensors detects a malfunction, an emergency message interrupts the normal display. This message appears for several seconds and then is turned off for a short period of time before recycling; it recycles continuously until the malfunction is corrected. The nine presently used emergency messages are:

- SERVICE BRAKE ON
- DOORS OPEN
- RADAR WARN—SLOW DOWN
- RESTRAINT SYSTEM OUT
- ANTI-SKID OUT
- BRAKE FLUID LOW
- OIL PRESSURE LOW
- WATER TEMP HIGH
- HAZARD

The system software determines the time for which the messages are displayed and then repeated, in addition to the priorities assigned to the individual messages in case there is more than one malfunction.

Conclusions

The introduction of microprocessors to the automobile opens many new exciting possibilities for improved performance, safety, and convenience in future cars. In this program we investigated the possibility of using microprocessors in two specific applications. One is the monitoring and display of a variety of performance- and safety-related sensors in the car—a fairly straightforward engineering task. The other, much more challenging, task involves adapting a noncooperative cw/fm

radar to automotive needs. The radar described here has successfully demonstrated the feasibility of the following functions: headway control with respect to other vehicles on the road; collision-mitigation braking when a collision is clearly imminent; and warning of obstacles and cars ahead.

In developing suitable algorithms for these functions, heavy emphasis was placed on ensuring that the radar system would be free from false alarms. This means the system must not respond to targets that are not in the direct path of the vehicle, such as parked cars on the shoulder, bridge abutments, overpasses, road signs, etc., or even to cars ahead, if there is still enough time for the driver to avoid them by evasive action. Thus, before radar-controlled braking is initiated, a number of special conditions have to be fulfilled—for example, car speed above 36 km/h, target within 25 m, and steering angle within certain limits dependent on car speed. This ensures that false targets do not lead to unnecessary braking, which in itself could lead to accidents that would cause the driver to lose faith in the system.

It is a major and very difficult task to determine an algorithm that can satisfy these safety considerations, yet still lessen the severity of accidents. Here, the microprocessor, when properly interfaced with a number of sensors, can perform a unique function, making radar-controlled braking a realistic possibility. With the limited amount of road-testing performed so far it appears that a radar system fulfilling the above requirements is indeed possible, although using much more refined algorithms and after more carefully

controlled road tests relating to the wide variety of possible accident situations.

Aside from the purely technical feasibility and public acceptance of such a system, there is one more very important factor to consider—cost. Initial estimates using the PRICE⁸ analysis indicate probable manufacturing costs of \$50 for the radar, \$90 for the microcomputer, and \$30 for the display. These numbers, although somewhat high, still show that such a system, with some further refinements, may become economically feasible in the next ten years.

Acknowledgments

The authors want to acknowledge many valuable contributions of R. Marx and E. McDermott in the assembly and testing of the system. Thanks are also due to R.J. Klensch, J. Schefer, D. Mawhinney, and W.C. Wilkinson for their helpful discussions during the radar development.

References

1. "A review of anti-skid braking," *Automotive Engineering*, Jul 1975, p. 34 (based on SAE papers 690213, 710248, and 741083.)
2. Robbi, A.D., and Tuska, J.W.; "Microcomputer-controlled spark advance system," RCA reprint RE-22-2-8.
3. Shefer, J.; Klensch, R.; Johnson, H.; and Kaplan, G.; "A new kind of radar for collision avoidance," SAE Automotive Engineering Congress and Exposition, Detroit, Mich., Feb 26, 1974.
4. Dolph, C.L.; "A current distribution for broadside arrays which optimizes the relationship between beam width and side-lobe level," *Proc. IRE*, Vol. 34 (Jun 1946) pp. 335-348.
5. Luck, D.G.C.; *Frequency Modulated Radar*, McGraw-Hill, New York, 1949.
6. Shefer, J.; Klensch, R.; Kaplan, G.; Johnson, H.; "Clutter-free radar for cars," *Wireless World* May 1974 pp. 117-122.
7. Lloyd, F.H., and Gerlough, D.L.; "A car-following simulation model," *Traffic Engineering and Control* May 1976 pp. 211-215.
8. Freiman, F.R.; *PRICE—A Parametric Cost Modeling Methodology*, RCA Government and Commercial Systems, Moorestown, N.J.

COSMAC Dictionary

Definitions of microprocessor and COSMAC-related terms
compiled by RCA Solid State Division

Access Time: Time between the instant that an address is sent to a memory and the instant that data returns. Since the access time to different locations (addresses) of the memory may be different, the access time specified in a memory device is the path which takes the longest time.

Accumulator: Register and related circuitry which holds one operand for arithmetic and logical operations. In the COSMAC Architecture, this is the D Register.

Additional Hardware: Microprocessor chips differ in number of additional ICs required to implement a functioning computer. Generally, timing, I/O control, buffering, and interrupt control require external components.

Address: A number used by the CPU to specify a location in memory.

Addressing Modes: See Memory Addressing Modes

ALU: Arithmetic-Logic Unit. That part of a CPU which executes adds, subtracts, shifts, AND's, OR's, etc.

Architecture: Organizational structure of a computing system, mainly referring to the CPU or microprocessor.

Assembler: Software that converts an assembly-language program into machine language. The assembler assigns locations in storage to successive instructions and replaces symbolic addresses by machine language equivalents. If the assembler runs on a computer other than that for which it creates the machine language, it is a **Cross-Assembler**.

Assembly Language: An English-like programming language which saves the programmer the trouble of remembering the bit patterns in each instruction; also relieves him of the necessity to keep track of locations of data and instructions in his program.

The assembler operates on a "one-for-one" basis in that each phrase of the language translates directly into a specific machine-language word, as contrasted with **High Level Language**.

Assembly Listing: A printed listing made by the assembler to document an assembly. It shows, line for line, how the assembler interpreted the assembly language program.

Asynchronous Operation: Circuit operation without reliance upon a common timing source. Each circuit operation is terminated (and next operation initiated) by a return signal from the destination denoting completion of an operation. (Contrast with **Synchronous Operation**).

Baud: A communications measure of serial data transmission rate; loosely, bits per second but includes character-framing START and STOP bits.

Benchmark Program: A sample program used to evaluate and compare computers. In general, two computers will not use the same number of instructions, memory words, or cycles to solve the same problem.

Bit: An abbreviation of "binary digit". (Single characters in a binary number.)

Bootstrap (Bootstrap Loader): Technique or device for loading first instructions (usually only a few words) of a routine into memory; then using these instructions to bring in the rest of the routine.

The bootstrap loader is usually entered manually or by pressing a special console key. COSMAC does not need one. See **Load Facility**.

Branch: See **Jump**.

Branch Instruction: A decision-making instruction which, on appropriate condition, forces a new address into the program counter. The conditions may be zero result, overflow on add, an external flag raised, etc. One of two alternate program segments in the memory are chosen, depending on the results obtained.

Breakpoint: A location specified by the user at which program execution (real or simulated) is to terminate. Used to aid in locating program errors.

Bus: A group of wires which allow memory, CPU, and I/O devices to exchange words.

Byte: A sequence of n bits operated upon as a unit is called an n-bit byte. The most frequent byte size is 8 bits.

Call Routine: See **Subroutine**

Clock: A device that sends out timing pulses to synchronize the actions of the computer.

Compiler: Software to convert a program in a high-level language such as FORTRAN into an assembly language or machine language program.

COSMAC: Generic description for the RCA family of compatible microprocessor products (1800 series). Based on a unique architecture, the COSMAC family includes CPU's, memories, I/O's, prototyping systems, and software.

COSMAC Development System: Microcomputer used for software development and system prototyping. Uses the COSMAC 1800 family microprocessor products.

COSMAC Software Development Package (CSDP): An assembler and interactive debugger/simulator for COSMAC microcomputer systems. The debugger is a powerful "software oscilloscope" that allows the user to start and stop the simulator, examine and modify the program variables, and dump and restore the entire simulated machine at will. CSDP is available either on national time keeping services or as a Fortran IV program that can be easily installed on a host computer.

Cross Assembler: A symbolic language translator that runs on one type of computer to produce machine code for another type of computer. See **Assembler**.

CPU (Central Processing Unit): That part of a computer system that controls the interpretation and execution of instructions. In general, the CPU contains the following elements:

- Arithmetic-Logic Unit (ALU)
- Timing and Control
- Accumulator
- Scratch-pad memory
- Program counter and address stack
- Instruction register and decode
- Parallel data and I/O bus
- Memory and I/O control

Cycle Stealing: A machine cycle stolen from the normal CPU operation for a DMA operation. See **DMA**.

Cycle Time: Time interval at which any set of operations is repeated regularly in the same sequence.

D Register: The accumulator in the COSMAC microprocessor.

Data Pointer: A register holding the memory address of the data (operand) to be used by an instruction. Thus the register "points" to the memory location of the data.

Data Register: Any register which holds data. In the COSMAC microprocessor, any one of the 16 x 16 scratch-pad registers can be used to hold two bytes of data.

Debug: To eliminate programming mistakes, including omissions, from a program.

Debug Programs: Debug programs help the programmer to find errors in his programs while they are running on the computer, and allow him to replace or patch instructions into (or out of) his program.

Designator: The three 4-bit registers P, X, and N in the COSMAC microprocessor are called designators. P and X are used to designate which one of the sixteen 16-bit

scratch-pad registers is used as the current program counter and the data pointer, respectively.

N can designate: one of the scratch-pad registers; an I/O device or command; a new value in P or X; and a further definition of an instruction.

Diagnostic programs: These programs check the various hardware parts of a system for proper operation; CPU diagnostics check the CPU, memory diagnostics check the memory, and so forth.

Direct Addressing: The address of an instruction or operand is completely specified in an instruction without reference to a base register or index register.

DMA: Direct Memory Access. A mechanism which allows an input/output device to take control of the CPU for one or more memory cycles, in order to write to or read from memory. The order of executing the program steps (instructions) remains unchanged.

Editor: As an aid in preparing source programs, certain programs have been developed that manipulate text material. These programs, called editors, text editors, or paper tape editors make it possible to compose assembly language programs on-line, or on a stand-alone system.

Evaluation Kit: PC board and associated components designed to allow the user to easily understand the system operation of a microprocessor.

Execute: The process of interpreting an instruction and performing the indicated operation(s).

Fetch: A process of addressing the memory and reading into the CPU the information word, or byte, stored at the addressed location. Most often, fetch refers to the reading out of an instruction from the memory.

Firmware: Software which is implemented in ROM's.

Fixed-instruction Computer (Stored-Instruction Computer): The instruction set of a computer is fixed by the manufacturer. The users will design application programs using this instruction set (in contrast to the **Micro-programmable Computer** for which the users must design their own instruction set and thus customize the computer for their needs.)

Fixed Memory: See **ROM**

Flag Lines: Inputs to a microprocessor controlled by I/O devices and tested by branch instructions.

Fortran: A high-level programming language generally for scientific use, expressed in algebraic notation. Short for "Formula Translator".

Guard: A mechanism to terminate program execution (real or simulated) upon access to data at a specified memory location. Used in debugging.

Hardware: Physical equipment forming a computer system.

Hexadecimal: Number system using 0, 1, . . . , A, B, C, D, E, F to represent all the possible values of a 4-bit digit. The decimal equivalent is 0 to 15. Two hexadecimal digits can be used to specify a byte.

High-Level Language: Programming language which generates machine codes from problem- or function-oriented statements. FORTRAN, COBOL, and BASIC are three commonly used high-level languages. A single functional statement may translate into a series of instructions or subroutines in machine language, in contrast to a low-level (assembly) language in which statements translate on a one-for-one basis.

Immediate Addressing: The method of addressing an instruction in which the operand is located in the instruction itself or in the memory location immediately following the instruction.

Immediate Data: Data which immediately follows an instruction in memory, and is used as an operand by that instruction.

Indexed Addressing: An addressing mode, in which the address part of an instruction is modified by the contents in an auxiliary (index) register during the execution of that instruction.

Index Register: A register which contains a quantity which may be used to modify memory address.

Indirect Addressing: A means of addressing in which the address of the operand is specified by an auxiliary register or memory location specified by the instruction rather than by bits in the instruction itself.

Input-Output (I/O): General term for the equipment used to communicate with a computer CPU; or the data involved in that communication.

Instruction: A set of bits that defines a computer operation, and is a basic command understood by the CPU. It may move data, do arithmetic and logic functions, control I/O devices, or make decisions as to which instruction to execute next.

Instruction Cycle: The process of fetching an instruction from memory and executing it.

Instruction Length: The number of words needed to store an instruction. It is one word in most computers, but some will use multiple words to form one instruction. Multiple-word instructions have different instruction execution times depending on the length of the instruction.

Instruction Repertoire: See Instruction Set

Instruction Set: The set of general-purpose instructions available with a given computer. In general, different machines have different instruction sets.

The number of instructions only partially indicates the quality of an instruction set. Some instructions may only be slightly different from one another; others rarely may be used. Instruction sets should be compared using benchmark programs typical of the application, to determine execution times, and memory requirements.

Instruction Time: The time required to fetch an instruction from memory and then execute it.

Interpreter: A program which fetches and executes "instructions" (pseudo instructions) written in a higher level language. The higher-level language program is a pseudo program. Contrast with Compiler.

Interrupt Request: A signal to the computer that temporarily suspends the normal sequence of a routine and transfers control to a special routine. Operation can be resumed from this point later. Ability to handle interrupts is very useful in communication applications where it allows the microprocessor to service many channels.

Interrupt Mask (Interrupt Enable): A mechanism which allows the program to specify whether or not interrupt requests will be accepted.

Interrupt Service Routine: A routine (program) to properly store away to the stack the present status of the machine in order to respond to an interrupt request; perform the "real work" required by the interrupt; restore the saved status of the machine; and then resume the operation of the interrupted program.

I/O Control Electronics (I/O Controller): The control electronics required to interface an I/O device to a computer CPU.

The powerfulness and usefulness of a CPU is very closely associated with the range of I/O devices which can be connected to it. One can not usually simply plug them into the CPU. The I/O Control Electronics will do the "matchmaking". The complexity and cost of the Control Electronics are very much determined by both the hardware and software I/O architecture of the CPU.

I/O Interface: See I/O Control Electronics

I/O Port: A connection to a CPU which is configured (or programmed) to provide a data path between the CPU and the external devices, such as keyboard, display, reader, etc. An I/O port of a microprocessor may be an input port or an output port, or it may be bidirectional.

Jump: A departure from the normal one-step incrementing of the program counter. By forcing a new value (address) into the program counter the next instruction can be fetched from an arbitrary location (either further ahead or back).

For example, a program jump can be used to go from the main program to a subroutine, from a subroutine back to the main program, or from the end of a short routine back to the beginning of the same routine to form a loop. See also the Branch Instruction. If you reached this point from Branch, you have executed a Jump. Now Return.

Linkage: See Subroutine

Load Facility: A hardware facility to allow program loading using DMA. It makes bootstrap unnecessary.

Loader: A program to read a program from an input device into RAM. May be part of a package of utility programs.

Loop: A self-contained series of instructions in which the last instruction can cause repetition of the series until a terminal condition is reached. Branch instructions are used to test the conditions in the loop to determine if the loop should be continued or terminated.

Low-Level Language: See Assembly Language

Machine: A term for a computer (of historical origin).

Machine Code: See Machine Language

Machine Cycle: The basic CPU cycle. In one machine cycle an address may be sent to memory and one word (data or instruction) read or written, or, in one machine cycle a fetched instruction can be executed. One machine cycle in the COSMAC microprocessor consists of eight clock pulses.

Machine Language: The numeric form of specifying instructions, ready for loading into memory and execution by the machine. This is the lowest-level language in which to write programs. The value of every bit in every instruction in the program must be specified (e.g., by giving a string of binary, octal, or hexadecimal digits for the contents of each word in memory).

Machine State: See State Code

Macro (Macroinstruction): A symbolic source language statement which is expanded by the assembler into one or more machine language instructions, relieving the programmer of having to write out frequently occurring instruction sequences.

Manufacturer's Support: It includes application information, software assistance, components for prototyping, availability of hardware in all configurations from chips to systems, and fast response to requests for engineering assistance.

Memory: That part of a computer which holds data and instructions. Each instructions or datum is assigned a unique address which is used by the CPU when fetching or storing the information.

Memory Address Register: The CPU register which holds the address of the memory location being accessed.

Memory Addressing Modes: The method of specifying the memory location of an operand. Common addressing modes are — direct, immediate, relative, indexed, and indirect. These modes are important factors in program efficiency.

Microcomputer: A computer whose CPU is a microprocessor. A microcomputer is an entire system with microprocessor, memory, and input-output controllers.

Microprocessor: Frequently called "a computer on a chip". The microprocessor is, in reality, a set of one, or a few, LSI circuits capable of performing the essential functions of a computer CPU.

Microprogrammable Computer: A computer in which the internal CPU control signal sequence for performing instructions are generated from a ROM. By changing the ROM contents, the instruction set can be changed. This contrasts with a Fixed-Instruction Computer in which the instruction set can not be readily changed.

Microterminal: Portable, hand-held terminal designed for controlling 1802-based systems like the Evaluation Kit.

Microtutor: Inexpensive microcomputer for first-level hands-on experience with microprocessor hardware and programming. Comes complete with CPU, memory, input and output devices, and power supply.

Mnemonics: Symbolic names or abbreviations for instructions, registers, memory locations, etc. A technique for improving the efficiency of the human memory.

Multiple Processing: Configuring two or more processors in a single system, operating out of a common memory. This arrangement permits execution of as many programs as there are processors.

Nesting: Subroutines which are called by subroutines are said to be nested. The nesting level is the number of times nesting can be repeated.

Nibble: A sequence of 4 bits operated upon as a unit. Also see Byte.

Object Program: Program which is the output of an automatic coding system, such as the assembler. Often the object program is a machine-language program ready for execution.

On-Line System: A system of I/O devices in which the operation of such devices is under the control of the CPU, and in which information reflecting current activity is introduced into the data processing or controlling system as soon as it occurs.

Op Code (Operation Code): A code that represents specific operations of an instruction.

Operating System: System software controlling the overall operation of a multi-purpose computer system, including such tasks as memory allocation, input and output distribution, interrupt processing, and job scheduling.

Page: A natural grouping of memory locations by higher-order address bits. In an 8-bit microprocessor, $2^8 = 256$ consecutive bytes often may constitute a page. Then words on the same page only differ in the lower-order 8 address bits.

PLA (Programmable Logic Array): A PLA is an array of logic elements which can be programmed to perform a specific logic function. In this sense, the array of logic elements can be as simple as a gate or as complex as a ROM. The array can be programmed (normally mask programmable) so that a given input combination produces a known output function.

Pointer: Registers in the CPU which contain memory addresses. See Program Counter and Data Pointer.

Program: A collection of instructions properly ordered to perform some particular task.

Program Counter: A CPU register which specifies the address of the next instruction to be fetched and executed. Normally it is incremented automatically each time an instruction is fetched.

PROM (Programmable Read-Only Memory): An integrated-circuit memory array that is manufactured with a pattern of either all logical zeros or ones and has a specific pattern written into it by the user by a special hardware programmer. Some PROMs, called EAROMs, Electrically Alterable Read-Only Memory, can be erased and reprogrammed.

Prototyping System: A hardware system used to breadboard a microprocessor-based product. Contains CPU, memory, basic I/O, power supply, switches and lamps, provisions for custom I/O controllers, memory expansion, and often, a utility program in fixed memory (ROM). See COSMAC Development System.

Pseudo Instruction: See Interpreter

Pseudo Program: See Interpreter

RAM (Random Access Memory): Any type of memory which has both read and write capability. It is randomly accessible in the sense that the time required to read from or to write into the memory, is independent of the location of the memory where data was most recently read from or written into. In contrast, in a **Serial Access Memory**, this time is variable.

Register: A fast-access circuit used to store bits or words in a CPU. Registers play a key role in CPU operations. In most applications, the efficiency of programs is related to the number of registers.

Relative Addressing: The address of the data referred to is the address given in the instruction plus some other number. The "other number" can be the address of the instruction, the address of the first location of the current memory page, or a number stored in a register. Relative addressing permits the machine to relocate a program or a block of data by changing only one number.

Resident Software: Assembler and editor programs incorporated with a prototyping system to aid in user program writing and development. See Software.

Return Routine: See Subroutine

ROM: Read-Only Memory (Fixed Memory) is any type of memory which cannot be readily rewritten; ROM requires a masking operation during production to permanently record program or data patterns in it. The information is stored on a permanent basis and used repetitively. Such storage is useful for programs or tables of data that remain fixed and is usually randomly accessible.

Routine: Usually refers to a sub-program, i.e., the task performed by the routine is less complex. A program may include routines. See Program.

Scratch-Pad Memory: RAM or registers which are used to store temporary intermediate results (data), or memory addresses (pointers).

Serial Memory (Serial Access Memory): Any type of memory in which the time required to read from or write into the memory is dependent on the location in the memory. This type of memory has to wait while undesired memory locations are accessed. Examples are paper tape, disc, magnetic tape, CCD, etc. In a Random Access Memory, access time is constant.

Simulators: Software simulators are sometimes used in the debug process to simulate the execution of machine-language programs using another computer (often a timesharing system). These simulators are especially useful if the actual computer is not available. They may facilitate the debugging by providing access to internal registers of the CPU which are not brought out to external pins in the hardware. See COSMAC Software Development Package.

Snapshots: Capture of the entire state of a machine (real or simulated) — memory contents, registers, flags, etc.

Software: Computer programs. Often used to denote general-purpose programs provided by the manufacturer, such as assembler, editor, compiler, etc.

Source Program: Computer program written in a language designed for ease of expression of a class of problems or procedures, by humans: symbolic or algebraic.

Stack: A sequence of registers and/or memory locations used in LIFO fashion (last-in-first-out). A **stack pointer** specifies the last-in entry (or where the next-in entry will go).

Stack Pointer: The counter, or register, used to address a stack in the memory. See Stack.

State Code: A coded indication of what state the CPU is — responding to an interrupt, servicing a DMA request, executing an I/O instruction, etc.

Subroutine: A subprogram (group of instructions) reached from more than one place in a main program. The process of passing control from the main program to a subroutine is a **subroutine call**, and the mechanism is a **subroutine linkage**. Often data or data addresses are made available by the main program to the subroutine. The process of returning control from subroutine to main program is **subroutine return**. The linkage automatically returns control to the original position in the main program or to another subroutine. See Nesting.

Subroutine Linkage: See Subroutine

Support: See Manufacturer's Support

Synchronous Operation: Use of a common timing source (clock) to time circuit or data transfer operations. (Contrast with **Asynchronous** operation)

Syntax: Formal structure. The rules governing sentence structure in a language, or statement structure in a language such as assembly language or Fortran.

Terminal: An Input-Output device at which data leaves or enters a computer system, e.g., teletype terminal, CRT terminal, etc.

Test and Branch: See Branch Instruction

Unbundling: Pricing certain types of software and services separately from the hardware.

Universal Asynchronous Receiver/Transmitter (UART): A device that translates serial data bits from two-wire lines to parallel format (receive mode) or parallel data bits to serial format for transmission over two-wire lines (transmit mode).

Utility Program: A program providing basic conveniences, such as capability for loading and saving programs, for observing and changing values in a computer, and for initiating program execution. The utility program eliminates the need for "re-inventing the wheel" every time a designer wants to perform a common function.

Word: The basic group of bits which is manipulated (read in, stored, added, read out, etc.) by the computer in a single step. Two types of word are used in every computer: Data Words and Instruction Words. Data words contain the information to be manipulated. Instruction words cause the computer to execute a particular operation.

Word Length: The number of bits in the computer word. The longer the word length, the greater the precision (number of significant digits). In general, the longer the word length, the richer the instruction set, and the more varied the addressing modes.

